DEVELOPMENT OF DATA ACQUISITION AND CONTROL FACILITIES FOR THE OPTIMIZATION OF DRIVE LINE EFFICIENCY

by

KENT DOUGLAS FUNK

B.S., KANSAS STATE UNIVERSITY, 1984

A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Mechanical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

Approved by:

J. Garth Thompson

Major Professor

PREFACE

The development of data acquisition and control facilities for the optimization of drive line efficiency has been a great challenge to my abilities. Although difficult, this work has allowed me to further develop my skills as both a programmer and an electronic hardware designer. Considerable personal satisfaction has been achieved by transforming the concepts discussed during project meetings into reality. I wish to thank all the individuals who have help me to make this project a success. In addition, a number of groups and individuals deserve special credits.

Thanks are extended to Caterpillar Tractor Company, Peoria IL, and Funk Manufacturing Company, Coffeyville KS, for their generous equipment donations. Likewise, thanks are also extended to the Kansas Department of Economic Development for providing matching funds to these equipment grants.

For their time and support, thanks go to the members of my graduate committee, Dr. Mark Schrock, Dr. Stanley Clark, and Dr. Ralph Turnquist. A special thanks goes to Dr. Garth Thompson, my major advisor, for allowing me a great deal of freedom in this work. His confidence and support are greatly appreciated.

Thanks also go to Mr. Mike Schwartz for his expertise in software design. Finally, thanks go to my wife, Tara, for her patience, support, and understanding.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Literature Review

Due to escalating fuel costs and increased capital costs associated with operating and owning agricultural tractors, considerable research has been conducted to improve the fuel efficiency and work rate of these units. The primary focus of this research has been to accurately define field load variations and to optimize engine power utilization. The potential savings from tractor performance optimization depends upon several factors; load variability, power level, engine characteristics, transmission characteristics, and tractive efficiency.

In a typical field application, the operator selects an appropriate gear ratio and throttle setting, and then allows the engine governor to compensate for load variability. The choice of gear and engine speed have traditionally been made by the operator, based upon field conditions and the type of work to be done. Recently, researchers have directed their efforts to improve the operator's decision. Meiring et. al. (1979) developed a tractor efficiency meter which determines the engine operating point in the speed and power range. By educating the operator to the characteristics of the engine speed vs. power map, the efficiency meter can be used as a valuable tool to improve both fuel consumption and productivity. Schrock et. al. (1982) developed a gear selection aid which informs the operator of an optimum gear and engine

speed for a given power demand. The average of four tests, with operators responding to instructions from the device, indicated a fuel savings of 19.8 percent compared with the operator's normal practice. There have been other similar works, but in each case the results are generally the same. Tractor efficiency can be improved by supplying the operator with additional information, and by educating the operator how to use that information.

Three limitations may be identified in past research work which will cause the optimization device to yield a non-optimal solution. The first limitation is that the operator must understand and respond to all information which is presented to him. If the operator does not understand, or if the device is continuously directing the operator to change the set point of the drive line, the device may be ignored. This problem can be avoided by automation of the drive line, thus taking the operator out of the optimization loop. Chancellor et. al. (1983) has developed a simple control device for tractor transmission ratio and governor setting so that experience could be gained in control design, operator interactions, and tractor performance. Early tests have shown good response, and work is continuing on development of a complete microprocessor based controller.

The second limitation is that the optimization device has a limited number of discrete transmission ratios from which to choose. This situation often results in a non-optimal solution. For example, the tractor may be operating at a specific gear ratio and throttle setting. First, the device examines the current state and establishes a base

point. Next, the device examines other gear ratio and throttle combinations which will produce the same power output. Due to the small number of admissible combinations which exist with conventional incremental transmissions, the device often can not improve the current combination. Obviously, the level of drive line optimization could be improved by increasing the number of admissible gear ratio and throttle setting combinations which will produce a given output power. This can be accomplished either by using a transmission with a large number of discrete ratios or a variable transmission with nearly an infinite number of ratios. One possible approach would be to use a hydrostatic unit, however, these units have a 10 to 15 percent lower efficiency than conventional gear type transmissions. These lower efficiencies would cancel much of the gain achieved by the optimization process. Another possible approach would be to use a mechanical continuously variable transmission (CVT). These transmissions have recently shown great promise due to improved reliability, durability, and efficiency. The major disadvantage of these units is that they suffer from a limited range of gear ratios, therefore, they must be cascaded with an additional discrete transmission in order to achieve the required ratio range.

The third limitation arises from an examination of the assumptions used in the development of most optimization algorithms. Generally, it has been assumed that the variation of transmission efficiency between gears is small and can be neglected in the selection of an optimum gear ratio and throttle setting. This has allowed researchers to develop algorithms based only upon engine optimization. Although this approach

often produces desirable results, extreme care must be exercised when developing algorithms which select the optimum combination from a large group of admissible gear ratios and throttle settings. Future work must move towards complete drive line optimization, rather than focusing only on the engine.

In summary of the current research work in the area of tractor performance optimization, most work has been done in the areas of defining load variability and optimization of engine efficiency. It is expected that future work will continue in the following areas.

1. Accurately define field load variations from which standard loading cycles can be developed for various field applications.

2. Develop algorithms and test devices which optimize work productivity and fuel economy over the entire tractor drive line.

3. Develop intelligent systems which adjust operating conditions based upon operator input, tractive efficiency, load variability, and other environmental conditions.

Project Overview

A joint study of Computer Control of Agricultural Tractor Drive Lines was initiated in April, 1984, between the Agricultural Engineering and Mechanical Engineering Departments at Kansas State University. The objective of this effort is to develop and test a computer control system for optimizing the performance of a diesel engine and a continuously variable transmission as applied in an agricultural tractor. The

objective of the project may be further divided into the following tasks.

1. Develop laboratory facilities for the study of drive line efficiency. This facility will include a drive line composed of a Caterpillar 3304 diesel engine, an experimental continuously variable transmission (CVT), a Funk model 2263 six speed power shift transmission, a Funk model 27 single speed planetary transmission, and a Midwest eddy current-type dynamometer. In addition, the facility will also include adequate computer facilities and instrumentation for implementation of data acquisition and control algorithms.

2. Collect and analyze data in order to determine performance relationships between control inputs and drive line outputs.

3. Develop an algorithm which will minimize fuel consumption at a specified work rate, and adequate controls in order to automate the optimization process.

4. Evaluate dynamic considerations of the control algorithm so that stable response is obtained from the controller.

5. Test and evaluate the performance of the algorithm against various loading cycles. These loading cycles will be derived from field data collected for the determination of field load variability.

This project is on-going with the test facility completed, the performance data collected, the basic optimization algorithm outlined, and the

standard loading cycles nearly completed.

Purpose Statement

The purpose of this thesis is to present the work completed on the development of computer facilities for implementation of data acquisition and control algorithms as related to the above project. These facilities consist of two separate computers; an ADAC 1000 data acquisition system based on the DEC LSI-11/23 microprocessor, and a Motorola MC68000 Educational Computer Board which is a single board computer based on the powerful MC68000 16-bit microprocessor.

CHAPTER II

COMPUTER ORGANIZATION

Project Needs

During the planning stages of the project, considerable attention was given to computer organization with respect to both data acquisition and control. Early discussions exposed three concerns. The first concern was that real time operation in data collection and control be achieved, secondly, that computer hardware should not be unnecessarily duplicated, and thirdly, that preliminary developments should not limit future work. As project discussions continued, the following list of specific needs emerged.

1. A supervisory system must be developed which has access to all data. It is the responsibility of the supervisor to insure complete system integrity. Desirable features of the supervisor are:
   - A. Periodically check all data against a set of boundary values in order to identify system abnormalities.
   - B. Provide a display of all data in an easy to read form along with appropriate warning messages.
   - C. Provide immediate system shut down in the case of catastrophic drive line failure.
2. A flexible data recording system must be developed which has access only to specific data. It is the responsibility of the data recorder to accurately measure and store data for later use in mapping the drive line and developing control algorithms. Desirable

features of the data recorder are:

A. Accurate control over sampling rates.

B. Flexibility in specifying which physical parameters are to be measured and recorded.

C. Data recorded should be stored in a readable form to facilitate spot checking.

3. A drive line controller must be developed which implements the optimization algorithm. It is the responsibility of the controller to adjust the state of the drive line based upon decisions made by the optimization algorithm. Desirable features of the controller are:

   A. Accurate real time operation.

   B. Control of the engine throttle position.

   C. Control of the CVT ring position which ultimately determines the CVT gear ratio.

   D. Control of the power shift gear ratio.

4. Additional drive line sub-system controllers must be developed as needed. These controllers should be developed and implemented independently of the drive line controller. Currently, the CVT oil temperature and the dynamometer loading pattern are the only sub-systems in need of computer control.

In order to meet these computer needs, a large amount of instrumentation has been developed to transform the physical drive line parameters into measurable signals. A complete list of parameters and their corresponding signal characteristics are presented in Appendix A. The information in Appendix A is divided into four sections: engine, CVT, power shift, and dynamometer. Each of the four main sections is further

divided into primary and secondary parameters.

Division of Responsibility

The above project needs are now divided as tasks between the two computers. The ADAC 1000 has responsibility for the supervisory functions, data recording, sub-system controls, and conversion of all analog data into digital forms. The Motorola single board computer has responsibility for drive line control and optimization. In addition, each of the computers has responsibility over digital data which is directly associated with their specific tasks.

It must be emphasized that the two computers can not work independently of each other. For example, the ADAC computer records certain digital information which only the single board computer can access. Therefore, an adequate communication link must be established between them. This links take the form of an RS-232 standard protocol, and all communication follows ASCII standards. These communications must be minimized since the inherent time delays will affect the real time operation of both computers.

In summary, this chapter has defined the project's overall computer needs, and has further grouped these needs into specific tasks for each of the two computers. In addition the need for inter-computer communications has been established and several concerns have been expressed. Chapter III will outline the work done on the ADAC 1000, and Chapter IV will present a discussion of the MC68000ECB.

CHAPTER III

ADAC 1000

System Evaluation

The ADAC 1000 data acquisltion system ls based upon the DEC LSI-11/23 microprocessor. It has been equipped with a 7.5 Mbyte hard dlsk, an 8 inch floppy disk, 256 Kbytes of RAM, and 4 serial ports. Data acquisltlon capabilities lnclude 32 channels of low levei analog inputs, 64 channels of high level analog inputs, 64 digital I/O ports, 4 analog output channels, and 4 pulse counters, along with a real time clock.

Snon after the division of responsibilities was made, a thorough evaluation of the ADAC 1000 was completed to determine the suitability of the system for its lntended tasks. This evaluation concluded that an extenslve update was in order to overcome previous reliability problems. The following modifications were made.

1. The computer enclosure was reorganized to improve access to data acqulsition modules and to allow for a pressurized air circulation system. In addition, a 1 KW Tripp Lite model SB-1000a UPS was added.

2. The interface between the data acquisition modules and real wnrld instrumentation was rebuilt. The new interface provides user access to all A/D, D/A, thermocouples, frequcncy, digital, and RS-232 signals. The interface also provides for custom signal

conditioning for a variety of applications.

3. The original DEC RT-11 operating system was replaced by 2.9BSD UNIX which is based on Bell Labs UNIX Version 7. In addition, all data acquisition service routines were rewritten in C, the intermediate language on which the UNIX operating system is based.

After the above modifications were made, extensive tests were conducted on both the operating system and data acquisition facilities. This work went well with few difficulties, and the complete system was ready for development of its specific tasks. Currently, the ADAC 1000 is a multi-user computing facility, which supports data acquisition capabilities, several compilers, graphics, statistical analysis, and many other 2.9BSD UNIX application programs. Since the modifications, work has proceeded with remarkable system reliability and the overall response has been excellent.

Multi-Process Approach

As outlined in chapter II, the ADAC 1000 is responsible for the supervisory functions, data recording, sub-system controls, and conversion of all analog data into digital forms. Due to the interactions between the various tasks, all of the ADAC responsibilities have been grouped together into one program. In addition, a terminal handler program has been developed which controls the communications between the real time task program and the terminal. The structure of these programs has been developed from concepts used in concurrent programming where inter-process communication is accomplished by message passing.

All together there are three processes executing under the supervision of the UNIX operating system. One of the three processes is the parent which initiates the creation of two other processes, known as children. These children are the screen and task processes. The parent also establishes all communication links between the children and itself. After successful creation of the two children and their intercommunications, the parent becomes the keyboard process. Following is a description of the relationships between the parent and the two children as shown in Figure 1, page 13.

The keyboard process accumulates input from the terminal keyboard until a full line of information has been recognized. After which, the keyboard process sends the line to the task process and informs the task process that information is waiting. In addition, the keyboard process echos input data to the screen process one character at a time.

One of the two children is a terminal screen process. This process accepts input from the task and keyboard processes, and displays the information on a static screen. The screen process input requires x-y coordinates, attributes, data format, and data. This process does not send information back to the other processes.

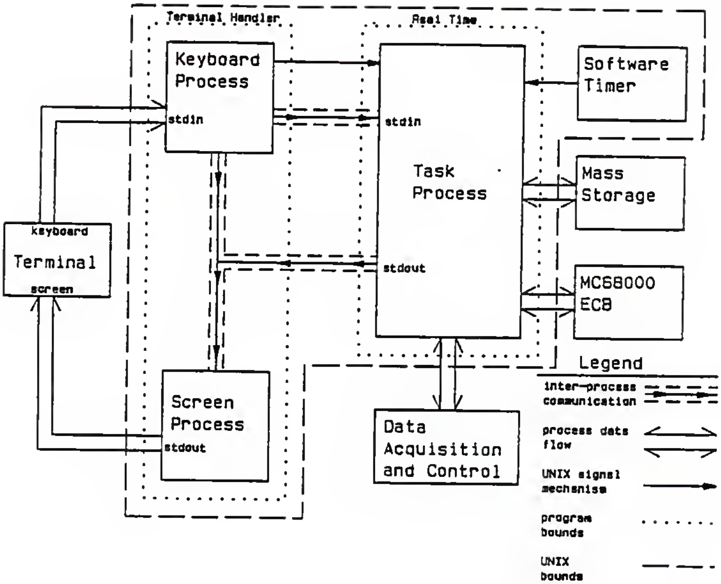


Figure 1 Multi-Process Relationships

The other child is the real time task process which carries out the specific tasks assigned to the ADAC 1000. When the parent process creates the task process, the standard input (stdin) is connected to the keyboard process and the standard output (stdout) is connected to the screen process. When the task process begins execution, the first step is to initialize all connections which have not been created by the parent. The initialization consists of five items.

1. Attach the UNIX signal associated with the keyboard process to the

task process.

2. Establish necessary links to the UNIX software timer in order to schedule real time activities.

3. Establish the communication link to the MC68000ECB.

4. Attach the data acquisition and control facilities to the UNIX operating system so that physical addressing is possible.

5. Send headings and other general information to the screen process.

After the task process initialization is complete, all relationships shown in Figure 1 have been created. It should be noted that the connection between the task process and mass storage is attached and detached as needed by normal execution of the task process.

Task Process Structure

As shown in Figure 2, page 16, the task process is divided into three regions; real time, human, and MC68000. These regions all have a source of external request stimulus, a common event detection mechanism, and a request handler. The common event mechanism detects the presence of an external request and then passes control to the appropriate request handler. In addition, all three regions are bound together by a common data structure which allows the regions to influence one another. Following is a detailed discussion of each of the three regions.

The real time region consists of the software timer request, event detection, real time handler, and the common data structure. During the

initialization of the task process, a link between the UNIX software timer and the common data structure was created. Each time the timer signal makes a request, a variable in the common data structure is changed. The event detection mechanism detects this change and passes control to the real time handler. It is the responsibility of the real time handler to schedule those tasks which require real time control. Within the real time handler, there are four real time loops.

1. The supervisory monitor loop reads all drive line parameters and compares these values to a set of boundary values. In addition, the monitor sends all parameter values along with warnings to the screen process.

2. The data recorder loop reads specific drive line parameters and stores these values in mass storage.

3. The CVT oil temperature controller loop reads the needed parameters to determine the necessary heater input control.

4. The dynamometer controller loop reads the needed parameters to determine the necessary dynamometer input control.

Each of these real time loops has a different loop time associated with them. Therefore, the real time handler must determine which loops need to be executed, if any, and passes control to those specific loops. It should be noted that the time interval of the software timer request is chosen as the greatest common factor of all four loop times.

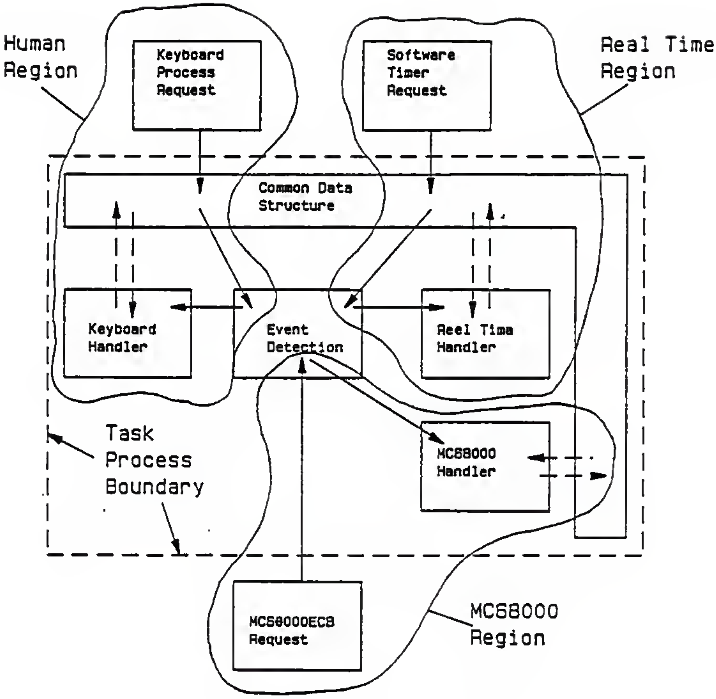


Figure 2 Task Process Structure

The human region consists of the keyboard process request, event detection, keyboard handler, and the common data structure. During the initialization of the task process, a link between the keyboard signal and the common data structure was created. Each time the keyboard process signals the task process that data is waiting, a variable in the common data structure is changed. The event mechanism detects this

change and passes control to the keyboard handler. It is the responsibility of the keyboard handler to read the waiting line of information, interpret meaning, and execute the command.

The keyboard handler provides a powerful mechanism by which the operator can examine and modify most of the common data structure. This ability allows the operator to control the normal execution of the other two regions. Although the possibilities are almost limitless, the basic capabilities of the keyboard handler can be grouped into four areas.

1. Data Acquisition.

   Through the keyboard hand[illegible] the operator can deal with all data acquisition facilities. This involves channel assignments to drive line parameters, calibration of offsets and gains, and printing additional information about a particular parameter to the screen.

2. Program Control.

   The keyboard handler also allows the operator to manipulate those variables in the common data structure which control program execution flow. This involves real time loop intervals, and the ability to turn specified loops on and off. In addition, parameters needed by the real time loops may be passed by the keyboard handler to the common data structure.

3. Boundary Values.

   As was described earlier, the supervisory monitor periodically compares all parameter values against a set of boundary values. These boundary values are stored in the common data structure, thus

allowing the keyboard handler to access them. This allows the operator to change the boundary values during normal execution of the task process.

4. Maintenance.

   The keyboard handler allows the operator to perform maintenance on the common data structure. This involves saving the entire structure on mass storage, allowing complete examination of the structure at a later time.

The program structure of the keyboard handler is much like a common interactive command line interpreter. However, all information needed to fully execute the command must be contained in a single line of information.

The MC68000 region consists of the MC68000ECB request, event detection, MC68000 handler, and the common data structure. During the initialization of the task process, a serial communications buffer was connected to the task process. Each time that the MC68000EC8 makes a request or sends data, characters appear at the communications buffer. The event detection mechanism reads characters in from the buffer until a line has been recognized and then verifies the line. After the event detection mechanism receives a valid request or data, control is passed to the MC68000 handler. This method of event detection is much different than that of the other two regions. In this case the event detection mechanism must poll the communication buffer since no UNIX signal is connected to the common data structure. It should be noted

that this form of event detection represents a violation of the real time intent of the task process. Therefore, communication between the task process and MC68000EC8 has been minimized.

Once control has been passed to the MC68000 handler, two possible actions can take place. If the request line contains input data, the request handler parses the line and stores the data in the common data structure. If the request line contains a request for data, the request handler reads the drive line parameters needed by the MC68000ECB and sends the data back through the serial line.

In summary, the purpose of this chapter is to give an overview of the work completed on the ADAC 1000. This work falls into two main categories; a complete system update, and development of the specific tasks. In order to implement the specific tasks, a great deal of software was needed. This software package is based upon concepts used in concurrent programming in order to preserve real time capabilities. A complete listing of the real time task process software is included in Appendix 8.

CHAPTER IV

MC68000ECB

System Overview

As outlined in chapter II, the responsibility of the MC68000ECB is to control the drive line in an optimal manner. In order to accomplish its intended tasks, the single board computer hardware has been greatly expanded. Software has also been developed to manage and test the added hardware. The purpose of this chapter is to discuss both hardware and software developments.

The MC68000ECB hardware has been expanded into a three board computer as shown in Figure 3, page 21. An enclosure has been built to provide a suitable environment for the MC68000ECB, the I/O expansion board, the high current driver board, and two power supplies. The main power supply services the MC68000ECB and the bus interface section of the I/O expansion board. The instrument power supply services the external digital inputs and the isolation section of the I/O expansion board. The high current driver board provides the connections between the driver interface section of the the I/O expansion board and the external high current digital outputs. Power for the digital outputs is provided by the engine electrical system. In addition, a number of manual switches provide control of the various power supplies. Finally, serial communications has been established between the MC68000ECB and its console, and between the MC68000ECB and the ADAC 1000.

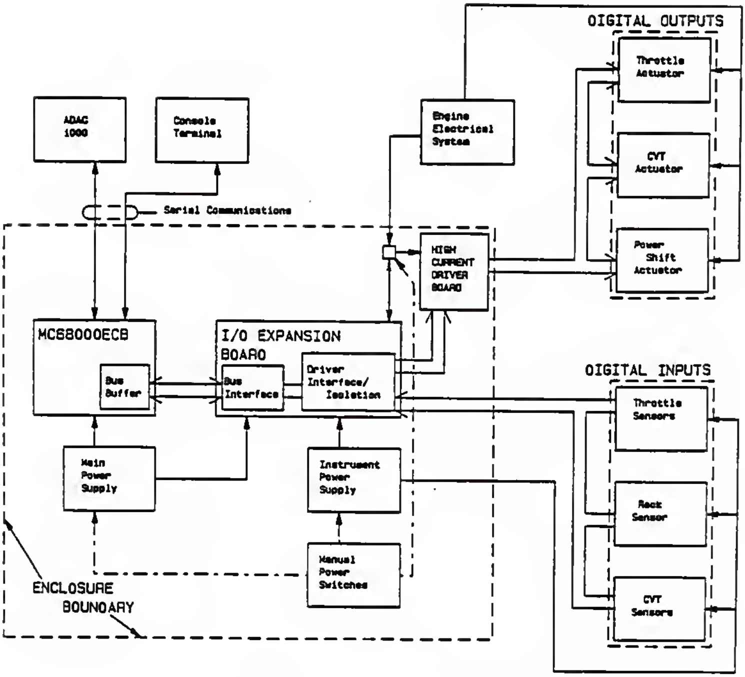


Figure 3 Hardware Configuration

The software developed on the MC68000ECB provides the framework for all future software developments. Since the final optimization algorithm has not been completely defined, a flexible software structure has been established. As shown in Figure 4, page 22, the software package is divided into four blocks; sequential program execution, software interrupt (SWI) processing, hardware interrupt (HWI) processing, and a common data structure.

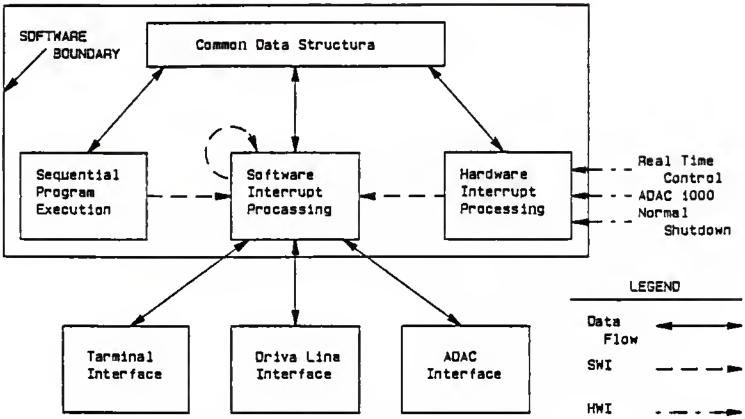


Figure 4 Software Structure

Normal sequential program execution begins by initialization of the common data structure, software and hardware interrupt vectors, and all system hardware. After the initialization sequence is complete, the entire software structure is functional. The normal program execution then enters a continuous loop. During one pass through the loop all inputs are read, the new desired drive line state is determined, and the outputs are set. In its final form, the sequential program execution block will contain the optimization algorithm. It is emphasized that normal program execution never deals directly with hardware devices. In order for sequential program execution to communicate with a hardware device, it must generate a software interrupt. The interrupt causes control to be passed to the software interrupt processing block which performs the hardware manipulation. All data transfer between the sequential program execution block and the software interrupt processing

block is accomplished through the common data structure.

The hardware interrupt processing block handles physically generated interrupts. These interrupts are non-sequential. That is, they may occur at any time without reguard to the current state of the microprocessor. These interrupts may occur from the hardware timer used for real time control, an ADAC 1000 request for data, or an operator request for system shutdown. In any case, the hardware interrupt processing block contains the code to handle these specific hardware interrupts. In order to service these interrupts, it is often necessary to communicate with external hardware devices. Therefore, the hardware interrupt processing block may also generate software interrupts in a similar manner as the sequential program execution block.

The software interrupt processing block is partitioned into a number of small groups of code called device drivers or utilities. Each of these small groups has a specific software interrupt level associated with it. Whenever a software interrupt is generated, the calling block must specify a level. The level of the interrupt specifies which utiiity or device driver is to be executed. This technique of control transfer is similar to cailing a subroutine except that the cailing program does not need to know the starting address of the desired utility.

The concept of software interrupt processing is the backbone of the entire software package. This has allowed individual device drivers and utilities to be written and tested before the other two blocks were completely defined. Since nearly all of the hardware is experimental, this

modular design has also increased the speed of testing hardware and software designs. After the basic software and hardware elements were proven, the rest of the system was built upon these basic elements.

The ahove discussion has given a brief overview of the entire system. Both hardware and software have been discussed in order to give prospective to the entire system development. In the following two sections, greater details will be discussed.

Hardware Details

The MC68000 Educational Computer Board (ECB) is a complete single board computer. Features of the computer are:

1. A 4 megahertz MC68000 16-bit MPU. This microprocessor has a 16-bit data bus and a 24-bit address bus. The address bus provides a memory addressing range of 16 megabytes. The processor also has eight 32-bit data registers, seven 32-bit address registers, two 32-bit stack pointers, a 32-bit program counter, and a 16-bit status register.

2. 32 Kbytes of dynamic RAM. Approximately 2 Kbytes are reserved for the operating system leaving 30 Kbytes for user programs.

3. 16 Kbytes of ROM. A small operating system called TUTOR resides on read only memory. The firmware provides monitor/debug, assembly/disassembly, program entry, and simple I/O control functions.

4. Two serial communication ports. These ports are fully RS-232C compatible and have hardware selectable baud rates.

5. A parallel port. This port can be used for a standard Centronics interface or custom I/O.

6. Audio tape serial I/O port. This feature allows for program storage on a standard tape recorder.

7. A 24-bit programmable timer. This timer is a synchronous device which can be used for generating or measuring both time delays and various frequencies. The timer can be clocked with a 5-bit prescaler or directly, and the clocking source can be the 4-Mhz system clock or an external clock.

8. Wire-wrap area provided for custom circuitry.

9. RESET and ABORT function switches.

A picture of the MC68000ECB and a functional block diagram are shown in Appendix C (pages 93 and 94). Additional information on the MC68000ECB is given in the Motorola User's Manual.

The first step in expanding the MC68000EC8 was to develop a standard asynchronous bus buffer. This buffer was installed in the wire wrap area on the board and provides full protection to all system data, address, and control lines which are taken off the board. Due to physical space limitations of the on board wire wrap area, only eight of the sixteen data lines are available. Otherwise, all address and control

lines are provided. This limitation is not serious since most peripherals developed for the MC68000 have an 8-bit data bus. Complete details of the buffer including parts list, board layout, and schematic are presented in Appendix C (pages 95 and 96).

Once the MC68000ECB system buses were buffered, they were extended onto a second board by a standard ribbon cable connection. The second board, called the I/O expansion board, provides access to all external devices. The bus interface section of the I/O expansion board consists of two MC68230 parallel interface/timers (PI/T) along with necessary address decoding. Each of the PI/Ts is actually two separate devices in one package. A parallel interface section provides two 8-bit external ports, and a timer section provides a 24-bit programmable timer. The driver interface and isolation section of the I/O expansion board consists of the hardware needed to provide the low current interface and high voltage isolation between the PI/Ts and the external devices. These facilities include two stepper motor drivers, two incremental encoder inputs, one absolute encoder input, and numerous single bit I/O. Complete schematics of the I/O expansion board are presented in Appendix C (pages 97 to 102). In addition, a schematic for the digital inputs is given on page 103, schematics for the high current outputs are presented on pages 104 to 108, and relevant data sheets are given in Appendix F.

The remaining hardware discussion will focus on two topics which are unique to the drive line interface. These discussions are intended to supplement the schematics in Appendix C. The first topic is optical isolation, and the second is interfacing stepper motors.

Optical Isolation

In this application, numerous external devices are connected to the I/O expansion board. Each of these devices has its own specific power supply requirements. It is important that these power supplies be separated in some way. Without separation, a failure in any one of the external branches could cause devastating damage to other devices and to the main boards. To provide complete isolation for all power supplies from each other and for all external devices from the main boards, a system of optical isolation has been developed.

Figure 5, shows a typical isolated output. The last chip in the low level board logic must have TTL Open Collector high current outputs. This allows the designer to customize the value of the pull-up resistor (R1) to the specific current requirements. The output of the open

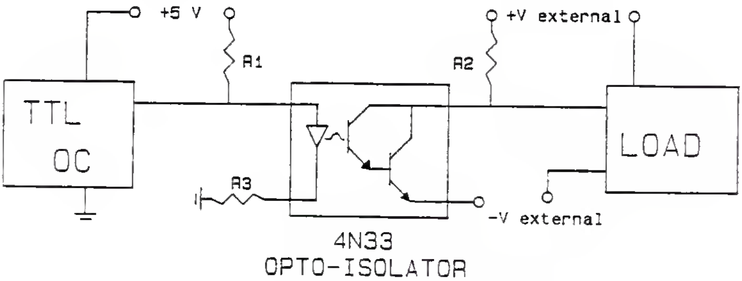


Figure 5 Typical Isolated Output

collector drives the input to a 4N33 Opto-Isolator. This chip provides full separation of power grounds, and 7500 V isolation. The input of

the 4N33 consists of an infrared light emitting diode and the output consists of a photo darlington transistor pair. The open collector design of the output of the 4N33 allows the designer to customize the value of the pull-up resistor (R2) which drives the input to the external load. Note that the load may need to include current amplifiers.

There are a number of design considerations in the selection of the pull-up resistors. The following is the procedure used in the present design.

1. R1 is selected so that the low level output current of the TTL open collector is not exceeded. The maximum value is typically 25 milliamps, therefore, R1 must be greater than 200 ohms. A value of 220 ohms has been used in the present design.

2. R2 is selected so the at the low level output current of the 4N33 is not exceeded. The maximum value is 50 milliamps, therefore, R2 must be greater than 100 ohms. The exact selection of the value of R2 varies with the characteristics of the load input current requirements.

3. The resistor R3 is used to improve the dynamic switching characteristics of the 4N33. This is because the diode has low input impedance and does not switch off cleanly after it has been driven on for an extended period of time. A value of 390 ohms was used in the present design. Note that if the value of R3 is too large, the forward current of the diode is diminished below the switch-on threshold.

An interesting side affect of the above circuit is that the logic is inverted. When the TTL output goes high, current is driven through the diode, turning the darlington pair on. With the transistors on, the output of thc 4N33 goes low, thus turning the load off. In contrast, when the TTL output goes low, the current through R1 sinks into the TTL output, thus turning the darlington pair off. With the transistors off, the output of the 4N33 goes high, thus turning the load on.

Although the above example demonstrates the use of opto-isolation for a single output, the basic concepts of the design can be used for inputs as well. In fact, entire groups of inputs and outputs can be isolated using this technique. In the present design, every input and output is fully isolated between the I/O expansion board and the rest of the external devices.

Interfacing Stepper Motors

Two dual phase unipolar stepper motors are used on the drive line. One motor controls the engine throttle position and the other controls the CVT ring position. Each of these motors has a stepper motor driver associated with it. Thc two drivers are functionally identical, however, each has been fine tuned to match the dynamics of their corresponding motors.

There are many tradeoffs between hardware and software in the design of a steppcr motor driver. In some systems, the stepper motor is driven with variable stepping rates in order to accelerate large loads. In other systems, a technique called micro stepping is used to improve

the accuracy of position control. In both of these cases, complete software control of the stepping sequence is required for the entire move. It is desirable, in this application, for software to play a minimal role in the process of moving the motors from one position to another. Since motor stepping rates are slow compared to the normal clocking speed of the microprocessor, constant supervision of a motor would waste valuable processing time. In addition, it would be difficult to control two motors moving at the same time using only one CPU. Therefore, a hardware system has been built and tested which requires the main control program to supply only two inputs. The program must calculate the desired number of steps and the relative direction of the move. In addition the software must enable the hardware to initiate the move. It is emphasized that the control software does not need to service the hardware at any time after the move is initiated. This allows the microprocessor to do other things while the stepper motors are moving.

Referring to Figure 6, page 31, each stepper driver consists of a timer, a clock source, a stepper driver chip, some additional logic, an optical coupler, and a high current driver section. The timer is part of the MC68230 PI/T and provides the necessary interface between the MC68000 system bus and the rest of the external hardware. The clock signal is derived from the system 1MHz system clock using decade counters. The stepper driver chip is a Motorola SAA1042 which generates the correct switching sequence logic for both full and half stepping modes. The output of the driver chip consists of four lines, one for

each coil of the stepper motor, which are first inverted and then passed through a series of 4N33 optical isolators. The outputs of the

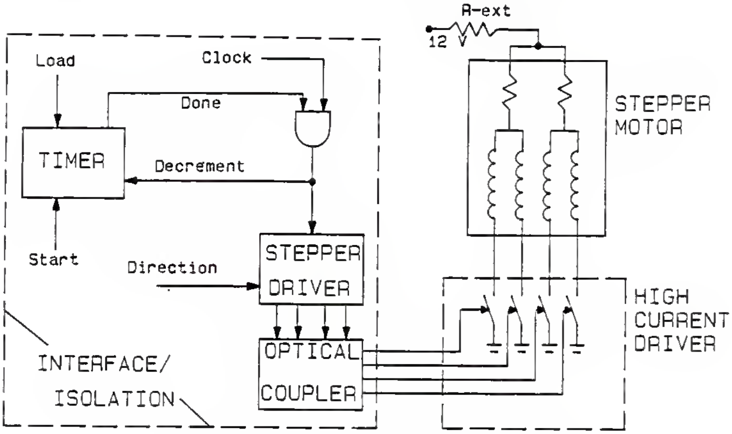


Figure 6 Stepper Motor Driver Circuit

optical isolators drive the bases of four SK3180 NPN Darlington power transistors. These transistors switch the four motor coils to ground. The stepper motors were originally designed to be driven with 4 to 5 volts DC, however, in this application it was more suitable to use the 12 volt power supply available at the engine electrical system. To do this an external resistor was placed in series with the stepper motor. The effect of this resistor is to develop a voltage divider with the input to the motor as the center node. By adjusting the value of the external resistance, the correct current is provided through the motor. In addition, this technique has the advantage of lowering the time constant (T= L/R) of the motor and improving its current switching

characteristics.

The basic operation of the hardware is as follows:

1. The number of desired steps is loaded into the timer, and the relative rotational direction is established at the stepper driver logic chip.

2. The timer is started allowing the "done" signal to go high. With the "done" signal high, the "clock" signal passes through the "AND" gate. The output of the "AND" gate provides a dual function of decrementing the counter and clocking the stepper driver logic chip. Note that the timer is decremented by one count for each cycle of the "decrement" signal and the stepper driver logic advances the stepper motor one step for each cycle of its input clock.

3. The circuit continues to decrement the timer and clock the stepper driver logic until the value in the timer goes to zero or "rolls over". At this point the timer drops the "done" signal low causing the "clock" signal to be gated off. With the clock signal gated off, the stepper driver logic is effectively halted without loss of the current sequence state.

During the initial design stages of the stepper motor drivers, it was anticipated that the open loop design would provide the needed accuracy and repeatability. However, during testing, the motors would occasionally loose steps. This was because the dynamic actuator loads were

greater than first anticipated. At this point each of the actuator gear trains were modified to include an optical incremental encoder. These encoders provide the repeatability necessary for data collection and control of the motors. Note that the lack of precision in the open loop design is not caused by a problem in the stepper motor driver, but rather is due to inadequate torque capacity of the stepper motors. In a final design, the feedback encoders would not be needed.

The output of the encoders is a pair of square wave pulse trains whose phase indicates relative direction. This type of output is known as "Quadrature" output. By correct separation of the two pulse trains, a suitable up/down counter can be used to keep track of the absolute position of the encoders. In addition, some reference must be established for the "home" or zero point of the counters. Two 20-bit counters were available (Spaulding, 1985) which correctly read quadrature output. These counters were added to the I/O expansion board and require seven control lines in order to multiplex the data to an 8-bit data bus. To accomodate the requirements of these devices, an 8-bit external data bus and an 8-bit external control bus were developed using one of the two off-board PI/Ts. The hardware specifications for the counters and the external buses are given in Appendix C (pages 99 to 103). The software specifications of the external buses are given in Appendix D (pages 142 and 143).

Software Details

As previously mentioned, software has been developed for the

MC68000ECB to support the hardware extensions. During the development of this software, a number of facilities were established in order to provide a suitable software development environment. Although the single board computer provides software development tools, some of these were found to be inadequate. The major deficiencies were program storage and program documentation.

The MC68000ECB provides a cassette recorder interface for program storage and retrieval, however, it is difficult to use and reliability is questionable. A more suitable solution was to use the hard disk storage on the ADAC 1000. Since the hardware communication was already established between the MC68000ECB and the ADAC 1000, all that was needed was the software support to upload and download programs across the serial communications line. Fortunately, the operating system for the MC68000ECB contains the necessary primitives to upload either S-records or memory dumps to a host computer and to download S-records from a host computer. With the available facilities, the only missing software was a communications handler for the ADAC 1000. This program, named "transfer", was relatively easy to write and has worked well. A complete listing of the transfer program is presented in Appendix E.

The other major deficiency, i.e. program documentation, was not so easily solved. The primary reason why documentation is so difficult on the MC68000ECB is that it is impossible to include comments in the source code written on the single board computer. This problem is best avoided by the use of a 68000 cross-assembler, unfortunately, they are expensive and none were locally available at the time. For this

application, the following procedure was used.

1. Type in the software on the MC68000ECB console using the single line assembler and keep accurate notes on paper.

2. Save the program segments in S-record format on the ADAC 1000 hard disk for future use.

3. Combine program segments into modules after each segment was debugged and save the modules in both S-record format and disassembled format on the ADAC 1000 hard disk.

4. Add comments to the disassembled modules from the development notes using an editor on the ADAC 1000.

In order to facilitate the last item above, a small comment editor was written. This line editor has the capability of appending long comments over many lines of assembly code and is intelligent about page formats. A complete listing of the comment editor is also included in Appendix E.

Although the above procedure has been used successfully to create all of the MC68000ECB software, one difficulty still exists. There is no way to automatically update a change in the documentation whenever a change is made in the source code. The only way to successfully modify the current software is to first make the changes on the MC68000ECB and accurately record those changes on scratch paper. Then upload the modification to the ADAC 1000 in S-record format for permanent storage. Finally, edit the original commented file and make the changes recorded on scratch paper. This can be a very difficult procedure which requires

the utmost attention to detail.

Once a procedure was established for development and documentation of software, attention was focused on the specific algorithms to be developed. As was previously discussed, there are four primary blocks of code; normal sequential execution, software interrupt processing, and hardware interrupt processing, all surrounded by a common data structure.

·The normal sequential execution block consists of a main routine and two subroutines, getdata and compute. The main routine provides the control for all normal sequential program execution, and currently executes an interactive environment (not real time) by which the user can manipulate the drive train. This configuration is useful for data mapping and system debugging. In its final form, the main routine will provide the real time control of the system. The getdata subroutine is called once each time through the main routine. The purpose of the getdata subroutine is to gather all necessary inputs and establish the current state of the drive line. The compute subroutine is also called once each time through the main routine. The purpose of the compute subroutine is to examine the current state of the drive line and compute a new desired state. In the current version, the compute subroutine is fully interactive so that the user inputs the new desired state. In its final form, the compute subroutine will implement the optimization algorithm. It should be noted that throughout the main routine and getdata subroutine, numerous software interrupts are generated in order to initialize and communicate with all system hardware. This creates an

advantage in software development since the hardware dependencies are all grouped together in the software interrupt processing block.

The hardware interrupt processing block is the least developed of the four primary blocks. In its final form, this block will be responsible for handling interrupts from three sources: ADAC 1000, real-time timer, and system shutdown requests. Currently only the ADAC 1000 interrupt is supported. The reason for not developing the additional facilities is that they are features of the final implementation and are not needed for data collection and debugging. Only one routine, HOST-INT, currently resides in the hardware interrupt proccssing block. Host-int provides the interrupt handler for all MC68000ECB to ADAC 1000 communications. In its final form, this routine will control communications in both directions. Currently, the routine only provides for data to be passed from the MC68000ECB to the ADAC 1000. This has proved sufficient for drive line mapping. Once the implementation of the control algorithm is in place, bi-directional data flow will be needed.

The software interrupt processing block provides the hardware interface to both other software blocks. This group of software has been fully developed in parallel with hardware developments and has been extensively tested. Following is a brief description of each routine in the software interrupt processing block.

1. P-INIT.

   P-init initializes the peripherals which are containcd on the MC68000ECB.

2. E-INIT.

   E-init initializes the peripherals which are contained on the I/O expansion bus.

3. SET-UP.

   Set-up provides an interactive mode of initializing the physical engine and transmissions. This procedure is executed only once and must be preformed after all computer hardware has been initialized.

4. MOVE-TH.

   Move-th provides the software interface to the stepper motor driver associated with the engine throttle.

5. MOVE-CVT.

   Move-cvt provides the software interface to the stepper motor driver associated with the CVT transmission.

6. SET-GEAR.

   Set-gear provides the software interface to the hardware associated with the Power Shift transmission.

7. TEST-STABLE.

   Test-stable insures that the drive line actuators are stable before releasing control back to normal sequential execution.

8. READ-RACK.

   Read-rack provides the software interface to the absolute encoder which is mechanically attached to the CAT diesel injector pump.

9. READ-THSTP.

   Read-thstp provides the software interface to the incremental encoder which is mechanically attached to the throttle stepper motor.

10. READ-FBSTP.

    Read-fbstp provides the software interface to the incremental encoder which is mechanically attached to the CVT ring drive.

11. REG-SAVE.

    Reg-save saves all CPU internal registers not saved as a part of normal context switching.

12. REG-RESTORE.

    Reg-restore restores all CPU internal registers which were saved hy reg-save.

The above software package is fully documented in Appendix D. Each of the routines is presented along with general discussions about common data, argument passing, and hardware specifications. Although all of the software is in assembly code, it is relatively easy to read since the code is filled with comments. In order to supplement Appendix D, a memory map (Table 1, page 40) has been specified. These memory specifications establish allocated space for all user software. The total memory available on the system is 32 Kbytes. The operating system (TUTOR) requires 1280 bytes of scratch and the RAM vector table requires 1024 bytes. Therefore a total of 30,464 bytes are available for user programs.

Table 1 MC68000ECB User Memory Allocation

Description	Address Range	# of 8ytes Available	# of 8ytes Used
Normal Sequential Execution			
Common Data	$0900 - $0DFF	1280	208
Main Routine			
Data	$0E00 - $0FFF	512	147
Text	$1000 - $11FF	512	289
Getdata Subroutine			
Data	$1200 - $12FF	256	0
Text	$1300 - $14FF	512	71
Compute Subroutine			
Data	$1500 - $1FFF	2816	49
Text	$2000 - $4FFF	12288	165
Software Interrupt Processing			
Common Data	$5000 - $507F	128	81
Local Data	$5080 - $57FF	1920	385
Text	$6000 - $6FFF	4096	1584
Hardware Interrupt Processing			
Local Data	$5800 - $5FFF	2048	45
Text	$7000 - $7FFF	4096	273
	Totals	30464	3297

In summary, the purpose of this chapter is to discuss the work completed on the MC68000ECB. This work includes both hardware expansions and software developments. A general overview was presented to give perspective to the entire system development, after which the hardware and software were discussed in more detail. During these discussions a number of appendices were cited which give specific details and comments about the hardware and software of the MC68000ECB.

SUMMARY

The objective of this study is to develop and test a computer control system for optimizing the performance of a diesel engine and a continuously variable transmission as applied in an agricultural tractor. In order to meet the objective, a number of tasks were defined.

1. Develop laboratory facilities for the study of drive line efficiency.

2. Collect and analyze data to determine performance relationships.

3. Develop an optimization and control algorithm for the drive line.

4. Evaluate dynamic considerations of the algorithm.

5. Test and evaluate the performance of the algorithm.

This project is on-going with the test facility completed, the performance data collected, and the basic optimization algorithm outlined. Plans for future work include: analyzing the collected data to estahlish relationships between control inputs and drive line outputs, and developing a computer simulation of the optimization algorithm in order to evaluate dynamic and performance considerations.

This thesis has presented the work completed on the development of computer facilities for implementation of data acquisition and control algorithms as related to the above study. In order to fulfill the

project's computer needs, two systems were developed. One system uses an ADAC 1000 data acquisition computer and is responsible for the supervisory functions, data recording, sub-system controls, and conversion of all analog data into digital forms. The other system uses a Motorola MC68000ECB single board computer and is responsible for drive line control and optimization.

The work completed on the ADAC 1000 falls into two main categories; a complete system upgrade, and development of a large software package. The structure of the software package is based upon concepts used in concurrent programming in order to preserve real time capabilities.

The work completed on the MC68000ECB includes both hardware and software developments. The hardware developments include bus expansion buffering, I/O expansion, optical isolation, and digital interfacing to external devices. The software developments provide the framework for all future developments. Currently, the software executes a fully interactive environment which is useful for drive line mapping.

REFERENCES

"Berkeley Unix Programmer's Manual," PDP-11 Version 2.9, Seventh Edition, Volume 1, Berkeley Software Division, Berkeley, CA, 1983.

Bourne, F. R., "The Unix System," Addison-Wesley Publishing, 1983.

Chancellor, W. J., and Thai, N. C., "Automatic Control of Tractor Engine Speed and Transmission Ratio," ASAE Paper No. 83-1061, Presented at ASAE Summer Meeting, Montana State University, Bozeman, June 1983.

Hancock, L., and Krieger, M., "The C Primer", 2nd ed., McGraw-Hill, New York, 1985.

Horowitz, P., and Hill, W., "The Art of Electronics," 3rd ed., Cambridge University Press, 1980.

Kelly-Bootle, S., and Fowler, B., "68000, 68010, and 68020 Primer," 1st ed., Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1985.

Kernighan, B. W., and Ritchie, D. M., "The C Programming Language," Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

"MC68000 16-Bit Microprocessor, Advanced Information," Motorola Inc., Publication ADI-814-R4, April 1983.

"MC68000 16-Bit Microprocessor, User's Manual," Motorola Inc., Third Edition, 1982.

"MC68000 Educational Computer Board, User's Manual," Motorola Inc., July 1982.

"MC68230 Parallel Interface/Timer, Advanced Information," Motorola Inc., Publication ADI-860-R2, Dec. 1983.

Meiring, P. and Lyne, P. W., "Power Demand Mapping of Tractor Operations," ASAE Paper No. 85-1050, Presented at ASAE Summer Meeting, Michigan State University, East Lansing, June 1985.

Meiring, P. and Rail, M. G., "An Operator Assist System to Optimize Tractor Efficiency," ASAE Paper No. 79-1048, Presented at the Summer Metting of ASAE and CSAE, University of Manitoba, Winnipeg, Canada, June 1979.

Morris, D. A., et. al., "Potential Fuel Savings Using a Tractor-Operator Feedback System," ASAE Paper No. 84-1077, Presented at ASAE Summer Meeting, University of Tennessee, Knoxville, June 1984.

Morris, W., editor, "The American Heritage Dictionary of the English Language," Houghton Mifflin Company, Boston, MA, 1976.

"Motorola Microprocessors Data Manual," Motorola Inc., Publication DL-120, 1981.

Schrock, M. D., Matteson, D. K., and Thompson, J. G., "A Gear Selection Aid for Agricultural Tractors," ASAE Paper No. 82-5515, Presented at ASAE Winter Meeting, Palmer House, Chicago, IL., Dec. 1982.

Smith, R. J., "Circuits, Devices, and Systems," 3rd ed., Wiley, New York, 1976.

Spaulding, G., "Instrumentation and Modification of the IRI/M50 Robot for Automatic Control Research," (unpubl M.S. Thesis, Kansas State University, 1985).

"The TTL Data Book for Design Engineers," 2nd ed., Texas Instruments Inc., 1976.

APPENDIX A

DRIVE LINE PARAMETERS

Primary Engine Parameters

1. Output Torque.
The engine output torque is measured by a Lebow Model 1228-10k in-line torque cell with a maximum rating of 10000 inch-pounds. The output signal is a millivolt level analog signal which is amplified and filtered by a Daytronic Model 3270 strain gage conditioner to achieve a 0 to 5 volt analog signal with 2 Hz. low pass characteristics. This signal is then digitized at the 1014 A/D board to give a final resolution of .975 ft-lb/bit.

2. Output Speed.
The engine output speed is measured by a 60-tooth gear in conjunction with a magnetic induction coil both of which are mounted to the above torque transducer. The output signal is a millivolt level analog signal which is amplified and filtered by a Oaytronic Model 3240 frequency conditioner to achieve a 0 to 5 volt analog signal with 2 Hz. low pass characteristics. This signal is then digitized on the 1014 A/D board to give a final resolution of 4.859 rpm/bit.

3. Governor Position.
The engine injector pump has been modified to accept a Litton 7NB10-5-S-1 absolute position encoder. The encoder converts the rotary motion of the pump "rack" to a standard TTL digital signal. Although the encoder gives 10 bits output, less than 90 degrees of rotation are achieved over the range of rack motion. Therefore, only eight bits are actually read by the control computer. The final resolution of the rack signal is 176 levels over the operating range of the rack.

4. Throttle Position.
The engine throttle position is controlled by a Sigma Model 20-2235D-28175 stepper motor. Control of the stepper motor is achieved through hardware on the MC68000ECB. In addition, a BEI Model L25G-100-ABZ-7400R-S incremental encoder has been incorporated into the throttle drive to give feedback of throttle position. The output of the encoder is in TTL digital quadrature format which is decoded through hardware on the MC68000ECB. Total control resolution of the throttle position is 1980 steps with a total of 3960 feedback levels over the range of throttle movement.

5. Throttle Home.
A single bit resolution on/off switch is also provided as part of the above throttle controller. The purpose of the this switch is to initialize the "home" or zero position of both the stepper motor and the feedback encoder.

6. Fuel Flow.
Engine fuel flow is measured gravimetrically. The fuel scale consists of an eight liter container suspended from an Amteck 8A-25-L8

load cell. The load cell is powered and conditioned to a high level analog signal by a Calex 166 bridge sensor. This signal is then digitized on the 1014 A/O board to give a final resolution of .008924 lbs-fuel/bit.

Secondary Engine Parameters

1. Oil Pressure. Engine oil pressure is measured by an Omega model PX-242-100G-5V pressure transducer. This signal is digitized on the 1014 A/O board to give a final resolution of .0977 psi/bit.

2. Exhaust Gas Temperature.
Engine exhaust gas temperature is measured by an Iron/Con type thermocouple.

3. Engine Coolant, Oil, and Ambient Air Temperatures.
These temperatures are measured by Copper/Con type thermocouples.

4. Fuel Temperature and API Number.
The engine fuel temperature is measured with an ordinary thermometer in degrees fahrenheit. The standard fuel API number is also collected by hand using a gravimetric bulb. This data is needed to convert fuel mass flow to volumetric flow. The data is entered into the data acquisition structure through the terminal keyboard.

5. Emergency Shut Off.
To provide for immediate system shut down in the case of catastrophic drive line failure, a throttle plate was installed in the engine intake system. Control of the emergency shut off is provide by a manual flip switch to a 110 VAC solenoid.

Primary CVT Parameters

1. Output Torque.
The CVT output torque is measured by a Lebow Model 1228-10k in-line torque cell with a maximum rating of 10000 inch-pounds. The output signal is a millivolt level analog signal which is amplified and filtered by a Daytronic Model 3270 strain gage conditioner to achieve a 0 to 5 volt analog signal with 2 Hz. low pass characteristics. This signal is then digitized at the 1014 A/O board to give a final resolution of .9735 ft-lb/bit.

2. Output Speed.
The CVT output speed is measured by a 60-tooth gear in conjunction with a magnetic induction coil both of which are mounted to the above torque transducer. The output signal is a millivolt level analog signal which is amplified and filtered by a Oaytronic Model 3240 frequency conditioner to achieve a 0 to 5 volt analog signal with 2 Hz. low pass characteristics. This signal is then digitized on the 1014 A/D board to give a final resolution of 4.8818 rpm/bit.

3. Ring Position.
The CVT ring position is controlled by a Sigma Model 21-4270D-200F03 stepper motor. Control of the stepper motor is achieved through hardware on the MC68000ECB. In addition, a BEI Model L25G-100-ABZ-7400R-S incremental encoder has been incorporated into the CVT ring drive to give feedback of ring position. The output of the encoder is in TTL digital quadrature format which is decoded through hardware on the MC68000ECB. Total control resolution of the ring position is 3400 steps with a total of 18891 feedback levels over the range of ring movement.

4. Ring Home.
A single bit resolution nn/off switch is also prnvided as part nf the above CVT ring controller. The purpose of the this switch is to initialize the "home" or zero position of both the stepper motor and the feedback encoder.

Secondary CVT Parameters

1. Oil Temperature Controller Inputs.
The CVT input oil temperature controller has a number of inputs. The following temperatures are measured:

   a. Input oil temperature.
   b. Exit oil temperature.
   c. Reservoir oil temperature.
   d. Heat exchanger oil input temperature.
   e. Tap water temperature.
   f. Heat exchanger water input temperature.
   g. Heat exchanger water exit temperature.

   All of these temperatures are measured by Copper/Con type thermocouples. In addition to the temperatures, the controller also requires an input oil temperature set point which is entered at the terminal keyboard.

2. Oil Temperature Controller Ouputs.
The controller has two outputs, each which sets the status of the water heater elements. The first element is controlled digitally as on/off, and the second element is controlled by the 1021 D/A board and a Johnson Model DQ-4100 solid state electric heat control unit. Total control output resolution is 2.44141 W/bit.

3. Oil Flow and Pressure.
The oil flnw and pressure are monitored to both branches of the oil loop. Pressure is measured by a standard dial pressure gauge, and flow is measured by two magnetic induction vane type flow meters. The flow meters output is a millivolt level analog signal. This signal is preconditioned to be read by the 1018 pulse counter board. The signal pulse trains are averaged over a one second time base. Conversion is then made to oil flow in gal/min.

Primary Power Shift Parameter

The power shift gear ratio is selected by appropriate engagement of six 12 VDC electro-hydraulic solenoids. Control of these solenoids is accomplished through a digital interface to the MC68000ECB. Software access of the current gear ratio is made through the communications link to the MC68000ECB.

Secondary Power Shift Parameters

1. Oil Pressure.
   Oil pressure is measured by a standard oil pressure dial gauge.

2. Oil Temperature.
   Oil temperature is measured by a Copper/Con type thermocouple.

Primary Dynamometer Parameters

1. Input Torque.
   The dynamometer input torque is measured by a Transducers Model T63H-200-C205 dual bridge load cell attached to 15.756 inch lever arm. One output of the load cell is used as feedback for the dynamometer controller. The other output is amplified and filtered by a Daytronic Model 3270 strain gage conditioner to achieve a 0 to 5 volt analog signal with 2 Hz. low pass characteristics. This signal is then digitized at the 1014 A/D board to give a final resolution of .6339 ft-lb/bit.

2. Input Speed.
   The dynamometer input speed is measured by a 60-tooth gear in conjunction with two magnetic induction coils. One of the pickups is used as feedback for the dynamometer controller. The other pickup is amplified and filtered by a Daytronic Model 3240 frequency conditioner to achieve a 0 to 5 volt analog signal with 2 Hz. low pass characteristics. This signal is then digitized on the 1013 A/D board to give a final resolution of 4.859 rpm/bit.

Secondary Dynamometer Parameters.

Both the input and output cooling water temperatures are measured using Copper/Con type thermocouples. These temperatures are used to provide dynamometer overloading protection.

Temperature Conversion Notes.

All of the above thermocouple outputs are digitized on the 1022 A/D board. Three types of thermocouple types are supported in the test lab. They are type T (Copper/Con), type J (Iron/Con), and type K (Chromel/Alumel). The following table has been established for temperature conversions.

Table 2 Temperature Conversions

TYPE	Sensitivity mv/bit	Regression Equation Temp(C)= (..........) (R^2= .9999+)	Valid Range deg C
T	0.00976	((.007727*mv-.436)*mv+24.91)*mv	0 to 400
J	0.0244	((-.0003982*mv-.0008997)*mv+18.52)*mv	0 to 750
K	0.0488	((.001582*mv-.09563)*mv+25.5)*mv	0 to 1360

APPENDIX B

ADAC 1000 SUPERVISOR SOFTWARE LISTINGS

```
************************************************************************
*
*
*    SUPPLEMENTAL LISTING FOR AOAC 1000 COMPUTER
*         CVT -- ENGINE PROJECT -- SUPERVISOR
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT O. FUNK
*    OATE:           4/5/85
*    FILE:           Makefile
*
************************************************************************


#This file contains the rules necessary to create
#    a program by the name "cvt".
#
#In order to run the program:
#  --> kmon cvt


# define global substitution variables
CC1= cc -c
CC2= cc -O -o
OBJECTS= main.o define.o keyboard.o doalarm.o mon.o cvtlinear.o
LIB= -ladac -lm


# define rules and dependencies for creation of object files
define.o : define.c control.h
     $(CC1) define.c
main.o : main.c control.h
     $(CC1) main.c
keyboard.o : keyboard.c control.h
     $(CC1) keyboard.c
doalarm.o : doalarm.c control.h
     $(CC1) doalarm.c
mon.o : mon.c control.h
     $(CC1) mon.c
cvtlinear.o : cvtlinear.c control.h
     $(CC1) cvtlinear.c


# define rules and dependencies for executables
cvt : $(OBJECTS)
     $(CC2) cvt $(OBJECTS) $(LIB)
```

```
***********************************************************************
*
*
*    C-SOURCE LISTING FOR ADAC 1000 COMPUTER
*         CVT -- ENGINE PROJECT -- SUPERVISOR
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT D. FUNK
*    OATE:           4/5/85
*    FILE:           control.h
*
***********************************************************************

/* Oeclare global variables used in main by other functions          */
#define NO68K
int  sigalarm();
int  sigkbdreq();
int  kbdflag, alarmflag;
int  fd68k;

/* definitions of attribute in scprint function                      */
#define   NORM          0
#define   UNDERLINE     1
#define   REVERSE       2

/* set up data file definitions                                      */
FILE *dfptr, *fopen();
char dfname[10];

/* set up structures for gman routine                                */
struct gv {         /* real variables                                */
     char gvname[7];
     double *gvad;
} gvs[];
struct flg {        /* integer variables                             */
     char flgname[7];
     int *flgad;
} flags[];

/* set up structure for high level data acquisition                  */
struct hlda {
     char parm[7];
     int chan, gain;
     double off, calcon;
     int a, b;
} hldax[];

/* set up structure for temperature data acquisition                 */
struct tmpda {
     char parm[7];
```

```
     int chan, type;
     int c, d;
} tmpdax[];

/* set up structure for pulse card data acquisition                        */
struct pulse {
     char parm[7];
     int chan;
     double tbase, const1, const2;
     int e, f;
} pulsex[];

/* set up structure for monitor subroutine limits                          */
struct limits {
     double *max, *warn;
} hl_lim[], tmp_lim[], pul_lim[];

/* set up space for command line                                           */
char keyword[5];
char name[10];
double data;

/* global variables declared                                               */
     /* engine parameters */
double ewtpmx;      /* engine water temp max                               */
double ewtpwn;      /* engine water temp warning                           */
double eegtmx;      /* exhuast gas temp max                                */
double eegtwn;      /* exhuast gas temp warning                            */
double erpmmx;      /* engine rpm max                                      */
duuble erpmwn;      /* enigne rpm warning                                  */
double etrkmx;      /* eng tork max                                        */
double etrkwn;      /* eng tork warning                                    */
double enopmn;      /* engine oil pressure min                             */
double enopwn;      /* engine oil pressure warning                         */
double enotwn;      /* engine oil temp warning                             */
double enotmx;      /* engine oil temp max                                 */
double enffmn;      /* wt of fuel in bucket > 2 lbs                        */
double api;         /* fuel api number                                     */
double ftmp;        /* fuel temperature                                    */
char point[10];     /* engine map referance point                          */

     /* cvt parameters */
double tcvtmx;      /* cvt oil temp exit mx used by monitor                */
double tcvtwn;      /* cvt oil temp exit warn used by monitor              */
double toomx;       /* cvt oil temp in max used by monitor                 */
double toomn;       /* cvt oil temp in min used by monitor                 */
double twimx;       /* max tmp of water in hx                              */
double twiwn;       /* warning tmp of water in hx                          */
double offmn;       /* cvt oil flow front min                              */
double offmx;       /* cvt oil flow front max                              */
double ofrmn;       /* cvt oil flow rear min                               */
```

```
double ofrmx;       /* cvt oil flow rear max                          */
double trpmmx;      /* cvt-pwrshift rpm max                           */
double trpmwn;      /* cvt-pwrshlft rpm warning                       */

     /* pwr-shift parameters */
double ttrkmx;      /* cvt-pwrshift tork max                          */
double ttrkwn;      /* cvt-pwrshift tork warning                      */
double psotmx;      /* pwr shlft oil temp max                         */
double psotwn;      /* pwr shift oil temp warning                     */

     /* final drive parameters */
double fdotmx;      /* final drive oil temp max                       */
double fdotwn;      /* final drive oil temp warning                   */

     /* dyno parameters */
double dwotmx;      /* dynomoter water temp out max                   */
double dwotwn;      /* dyno water temp out warning                    */
double drpmmx;      /* dyno rpm max                                   */
double drpmwn;      /* dyno rpm warning                               */
double dtrkmx;      /* dyno trk max                                   */
double dtrkwn;      /* dyno trk warn                                  */

     /* ambient parameters */
double alrthl;      /* ambient air temp hi                            */
double airtlo;      /* ambient air temp lo                            */

     /* program control flags */
int timint;         /* alarm interupt time (sec)                      */
int montim;         /* monitor loop time (sec)                        */
int dyctim;         /* dyno loop time (sec)                           */
int hrctlm;         /* heater control loop time (sec)                 */
int dattim;         /* data collection loop time (sec)                */
int datmax;         /* data collectlon interval (number of samples)   */
int monflg;         /* monitor enable                                 */
int dycflg;         /* dyno enable                                    */
int hrcflg;         /* heater enable                                  */
int datflg;         /* data loop enable                               */
int orphan;         /* kmon exiting flag                              */
int oniine;         /* m68k online flag                               */
int waitmx;         /* wait state for request call on 68k             */
int daval;          /* temporary value for D/A in heater              */
int ttlval;         /* temporary value for TTL in heater              */
     /* m68k variables */
int req;            /* control flag for request                       */
int rack;           /* rack reading                                   */
int thstp;          /* throttle step reading                          */
int rgstp;          /* ring step reading                              */
int gear;           /* PS gear                                        */
int fbstp;          /* cvt feedback reading                           */
     /* heater control input */
double hset;   /* cvt oll temp in as set point for heater control     */
```

```
************************************************************************
*
*
*    C-SOURCE LISTING FOR ADAC 1000 COMPUTER
*         CVT -- ENGINE PROJECT -- SUPERVISOR
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT D. FUNK
*    DATE:          4/5/85
*    FILE:          define.c
*
************************************************************************


/* This file contains all compile definitions of global variables     */
#include <stdio.h>
#include "control.h"
#include <adac.h>

/* define data file default                                            */
char dfname[10]= "d.tmp";

/* global variables initialized                                        */
     /* engine parameters */
double ewtpmx=  105.0;   /* engine water temp max                      */
double ewtpwn=   93.0;   /* engine water temp warning                  */
double eegtmx=  705.0;   /* exhuast gas temp max                       */
double eegtwn=  650.0;   /* exhuast gas temp warning                   */
double erpmmx= 2500.0;   /* engine rpm max                             */
double erpmwn= 2300.0;   /* enigne rpm warning                         */
double etrkmx=  240.0;   /* eng tork max                               */
double etrkwn=  260.0;   /* eng tork warning                           */
double enopmn=   30.0;   /* engine oil pressure min                    */
double enopwn=   40.0;   /* engine oil pressure warning                */
double enotwn=   93.0;   /* engine oil temp warn                       */
double enotmx=  105.0;   /* engine oil temp max                        */
double enffmn=    5.0;   /* minimum fuel in the fuel bucket            */

     /* cvt parameters */
double tcvtmx=   75.0;   /* cvt oil temp exit mx used by monitor       */
double tcvtwn=   65.0;   /* cvt oil temp exit warn used by monitor     */
double toomx=     2.0;   /* cvt oil temp in cvt, max used by monitor   */
double toomn=     2.0;   /* cvt oil temp in cvt, min used by monitor   */
double twimx=    65.0;   /* water temp in max                          */
double twiwn=    60.0;   /* warning water in tmp                       */
double offmn=     4.5;   /* cvt oil flow front min                     */
double offmx=     5.5;   /* cvt oil flow front max                     */
double ofrmn=     4.5;   /* cvt oil flow rear min                      */
double ofrmx=     5.5;   /* cvt oil flow rear max                      */
double trpmmx= 1750.0;   /* cvt-pwrshift rpm max                       */
```

```
double trpmwn= 1610.0;   /* cvt-pwrshift rpm warning                      */

     /* pwr-shift parameters */
double ttrkmx=  667.0;   /* cvt-pwrshift tork max                         */
double ttrkwn=  615.0;   /* cvt-pwrshift tork warning                     */
double psotmx=  100.0;   /* pwr shift oil temp max                        */
double psotwn=   90.0;   /* pwr shift oil temp warning                    */

     /* final drive parameters */
double fdotmx=  100.0;   /* final drive oil temp max                      */
double fdotwn=   90.0;   /* final drive oil temp warning                  */

     /* dyno parameters */
double dwotmx=   45.0;   /* dynomoter water temp out max                  */
double dwotwn=   40.0;   /* dyno water temp out warning                   */
double drpmmx= 4000.0;   /* dyno rpm max                                  */
double drpmwn= 3800.0;   /* dyno rpm warning                              */
double dtrkmx=  190.0;   /* dyno trk max                                  */
double dtrkwn=  175.0;   /* dyno trk warn                                 */

     /* ambient parameters */
double airthi=   32.0;   /* ambient air temp hi                           */
double airtlo=   22.0;   /* ambient air temp lo                           */

     /* program control flags */
int timint=        1;    /* number of seconds between intcrupts           */
int montim=       10;    /* monitor loop time in seconds                  */
int dyctim=       10;    /* dyno loop time in seconds                     */
int hrctim=        5;    /* heater loop time in seconds                   */
int dattim=       10;    /* data collection loop time in seconds          */
int datmax=       18;    /* maximum number of data sets                   */
int monflg=        1;    /* monitor loop enable                           */
int dycflg=        0;    /* dyno loop enable                              */
int hrcflg=        0;    /* heater loop enable                            */
int datflg=        0;    /* data loop enable                              */
int daval=         0;    /* heater D/A value                              */
int ttlval=        0;    /* heater TTL value                              */
int orphan=        0;    /* flag set when CVT is backgrounded             */
int online=        0;    /* initially m86k is offline                     */
int waitmx=        2;    /* wait state for data request on 68k            */


/* initialize the structure gv                                            */
struct gv gvs[] {
     "ewtpmx", &ewtpmx,  /* real variable name and storage address        */
     "ewtpwn", &ewtpwn,
     "eegtmx", &eegtmx,
     "eegtwn", &eegtwn,
     "erpmmx", &erpmmx,
     "erpmwn", &erpmwn,
     "etrkmx", &etrkmx,
```

```
     "etrkwn", &etrkwn,
     "enopmn", &enopmn,
     "enopwn", &enopwn,
     "enotwn", &enotwn,
     "enotmx", &enotmx,
     "enffmn", &enffmn,
     "tcvtmx", &tcvtmx,
     "tcvtwn", &tcvtwn,
     "toomx",  &toomx,
     "toomn",  &toomn,
     "hset",   &hset,
     "twimx",  &twimx,
     "twiwn",  &twiwn,
     "offmn",  &offmn,
     "offmx",  &offmx,
     "ofrmn",  &ofrmn,
     "ofrmx",  &ofrmx,
     "trpmmx", &trpmmx,
     "trpmwn", &trpmwn,
     "ttrkmx", &ttrkmx,
     "ttrkwn", &ttrkwn,
     "psotmx", &psotmx,
     "psotwn", &psotwn,
     "fdotmx", &fdotmx,
     "fdotwn", &fdotwn,
     "dwotmx", &dwotmx,
     "dwotwn", &dwotwn,
     "drpmmx", &drpmmx,
     "drpmwn", &drpmwn,
     "dtrkmx", &dtrkmx,
     "dtrkwn", &dtrkwn,
     "airthi", &airthi,
     "airtlo", &airtlo,
     "",                 /* error check at end of structure          */
};


/* Initialize the integer variable structure                         */
struct flg flags[] {
     "timint", &timint,  /* int variable name and address            */
     "montim", &montim,
     "dyctim", &dyctim,
     "hrctim", &hrctim,
     "dattim", &dattim,
     "datmax", &datmax,
     "monflg", &monflg,
     "dycflg", &dycflg,
     "hrcflg", &hrcflg,
     "datflg", &datflg,
     "daval",  &daval,
     "ttlval", &ttlval,
```

```
     "online", &online,
     "waitmx", &waitmx,
     "",
};


/* Initialize high level data acquistion structure                          */
struct hlda hidax[] {
     "erpm", 26, 1,  -2.0, 4.859086,  8,  4, /* engine rpm                  */
     "etrk", 28, 1,   1.0, 0.975161,  8,  5, /* engine tork                 */
     "trpm", 27, 1,   1.0, 4.881813, 23,  4, /* cvt-trans rpm               */
     "ttrk", 29, 1,   1.0, 0.973510, 23,  5, /* cvt-trans tork              */
     "drpm", 34, 1,  -1.0, 4.859086, 55,  4, /* dyno rpm                    */
     "dtrk", 32, 1,  -2.0, 0.482775, 55,  5, /* dyno tork                   */
     "enop", 16, 1, 209.0, 0.097704,  8, 10, /* engine oil pressure         */
     "enff", 25, 1, 277.0, 0.008924,  8,  6, /* engine fuel flow            */
     "",                          /* error check at end of structure        */
};

/* initialize temp data acquistion structure                                */
struct tmpda tmpdax[] {
     "ewtp", 20, 3,  8,  8,   /* engine water tmp                           */
     "eegt", 30, 0,  8,  7,   /* engine exhaust gas tmp                     */
     "enot", 21, 3,  8,  9,   /* engine oil temp                            */
     "tcvt", 26, 3, 23,  8,   /* cvt oil tmp exit                           */
     "twi",  29, 3, 23,  6,   /* water temp of hx loop                      */
     "psot", 23, 3, 39,  4,   /* pwr shift oil tmp                          */
     "fdot", 22, 3, 39,  5,   /* final drive oil tmp                        */
     "dwot", 17, 3, 55,  7,   /* dyno water outlet temp                     */
     "too",  27, 3, 23,  7,   /* cvt oil tmp in                             */
     "airt", 19, 3, 39,  7,   /* ambient air tmp                            */
     "ttnk", 25, 3, 23,  9,   /* cvt oil tmp @ tank                         */
     "toi",  24, 3, 23, 10,   /* cvt oil tmp at hx                          */
     "dwit", 18, 3, 55,  6,   /* dynot water inlet temp                     */
     "two",  28, 3, 23, 11,   /* cvt water temp exit                        */
     "",                      /* error checking at end of structure         */
};


/* initialize pulse card data acquistion structure                          */
struct pulse pulsex[] {
     "off", 0, 1.0, 1.414, 5.572, 23, 12,    /* cvt oil flow front          */
     "ofr", 1, 1.0, 1.05,  7.777, 23, 13,    /* cvt oil flow rear           */
     "",                      /* error check at end of structure            */
};


/* initialize monitor limits structure                                      */
struct limits hl_lim[] {
     &erpmmx, &erpmwn,   /* max-warn storage addresses => struct hlda */
     &etrkmx, &etrkwn,
```

```
     &trpmmx, &trpmwn,
     &ttrkmx, &ttrkwn,
     &drpmmx, &drpmwn,
     &dtrkmx, &dtrkwn,
     &enopmn, &enopwn,
};

struct limits tmp_lim[] {
     &ewtpmx, &ewtpwn,   /* max-warn storage addresses => tmpda       */
     &eegtmx, &eegtwn,
     &enotmx, &enotwn,
     &tcvtmx, &tcvtwn,
     &twimx,  &twiwn,
     &psotmx, &psotwn,
     &fdotmx, &fdotwn,
     &dwotmx, &dwotwn,
     &toomx,  &toomn,
     &airthi, &airtlo,
};

struct limits pul_lim[] {
     &offmx, &offmn,     /* max-min storage addresses => struct pulse */
     &ofrmx, &ofrmn,
};


/* initialize m68k variables                                          */
int req=       0;
int rack=      0;
int thstp=     0;
int rgstp=     0;
int gear=      0;
int fbsto=     0;

/* heater control input                                               */
double hset=  65.0; /* cvt oil temp-in set point for heater control   */
```

```
************************************************************************
*
*
*    C-SOURCE LISTING FOR ADAC 1000 COMPUTER
*         CVT -- ENGINE PROJECT -- SUPERVISOR
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT D. FUNK
*    DATE:           4/5/85
*    FILE:           main.c
*
************************************************************************

#include <signal.h>
#include <stdio.h>
#include "control.h"
#include <adac.h>

/* this function is used to handle unix alarm interuputs              */
sigalarm() {
     signal(SIGALRM, sigalarm);
     alarm(timint);
     alarmflag++;
}

/* this function handles the keyboard process interupt                */
sigkbdreq() {
     signal(SIGINT, sigkbdreq);
     kbdflag++;
}


/******                    MAIN()                              ******/
main(argc, argv)
int  argc;
char **argv;
{
     int err;

     signal(SIGQUIT,SIG_IGN);
     signal(SIGALRM, sigalarm);
     signal(SIGINT,sigkbdreq);

     /* open up the 68k line here */
     if ((fd68k = open("/dev/ttyd0", 2)) < 0) {
          fprintf(stderr. "cannot open 68k line\n");
          fprintf(stderr, "   cvt exiting\n");
          exit(1);
     }
```

```
     alarm(timint);
     adacinit();
     paint();

loop:
     err= do68k(1, 0);
     while(alarmflag > 0)
          doalarm();
     if(kbdflag)
          keyboard();
     goto loop;
}


/******                      68k handler                        ******/
do68k(entry, option)
int entry, option;
{
     static int first= 0;
     int error, ret, num;
     char rdbuf[80];
     char *lptr;

     if (first == 0) {
          *rdbuf= ' ';
          ret= 0;
          first= 1;
     }

     error= 0;
     if (online == 0) {
          error= 1; /* 68k is not here now */
          goto not_here;
     }

     switch(entry) {
          case 1:          /* entry point for main */
               ret= read(fd68k, rdbuf, 80);
               if (ret > 0) {
                    rdbuf[ret]= ' ';

                    switch(*rdbuf) {
                         case 'R':
                              service();
                              break;
                         case 'S':
                              break;
                         default:
                              ;
                    }
```

```
                *rdbuf= ' ';
                ret= 0;
            }
            break;

        case 2:         /* entry point for monitor */
            if (req) {
                goto not_here;
            }
            if (option == 1) {
                write(fd68k, "R", 1);
                break;
            }
            if (option == 2) {
                ret= read(fd68k, rdbuf, 80);
                if (ret > 0) {
                    rdbuf[ret]= ' ';
scprint(1, 1, 0,"                                                   ");
                    scprint(1, 1, 0, "%s", rdbuf);
                    moveto(99,99);
                    if (*rdbuf != 'S') {
                        error= 5;
                        break;
                    }
                    lptr= rdbuf + 1;
                    num= sscanf(lptr, "%d %d %d %d %d",
                                &rack, &thstp,
                                &rgstp, &fbstp,
                                &gear);
                    if (num != 5) {
                        error= 4;
                    }
                    *rdbuf= ' ';
                    ret= 0;
                    break;
                    }
                error= 3;
            }
            break;

        case 3:         /* entry point for data routines */
            if (option == 1) {
                write(fd68k, "R", 1);
                break;
            }
            if (option == 2) {
                ret= read(fd68k, rdbuf, 80);
                if (ret > 0) {
                    rdbuf[ret]= ' ';
scprint(1, 1, 0,"                                                   ");
                    scprint(1, 1, 0, "%s", rdbuf);
```

```
                        moveto(99,99);
                        if (*rdbuf != 'S') {
                             error= 5;
                             break;
                        }
                        lptr= rdbuf + 1;
                        num= sscanf(lptr, "%d %d %d %d %d",
                                       &rack, &thstp,
                                       &rgstp, &fbstp,
                                       &gear);
                        if (num != 5) {
                             error= 4;
                        }
                        *rdbuf= ' ';
                        ret= 0;
                        break;
                   }
                   error= 3;
              }
              break;

         default:
              error= 2;
    }

not_here:
    return(error);
}


/******                 Handle a 68k request                ******/
service()
{
    /* provide 68k data services here */
         /* Not Developed */
    ;
}


/*
do68k() error handling.

general:
0 == normal
1 == 68k not here
2 ==  entry option error for do68k call

specific by entry and option:
1 x
    none
```

```
2 1
    none
2 2
    3= new line not here yet (data not current)
    4= wrong number of arguements in new line.
    5= 'S' tag missing on new line.
3 1
    none
3 2
    3= new line not here yet (data not current)
    4= wrong number of arguements in new line.
    5= 'S' tag missing on new line.
*/
```

```
************************************************************************
*
*
*     C-SOURCE LISTING FOR ADAC 1000 COMPUTER
*          CVT -- ENGINE PROJECT -- SUPERVISOR
*
*     DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*     AUTHOR:         KENT D. FUNK
*     DATE:           4/5/85
*     FILE:           keyboard.c
*
************************************************************************

#include <stdio.h>
#include "control.h"


/******                        Keyboard()                          ******/
/*
Keyboard service routine.
     Purpose: Interpret the first letter of the keyword and direct
          program control to the proper routine. */
keyboard()
{
     char line[80];
     char *ptr;
     char *nptr;
     int n;

     kbdflag--;          /* negates calling condition */
     fgets(line, 80, stdin);
     if (feof(stdin))
          {
          printf("exiting gracefully\n");
          exit(0);
          }
     if (strcmp(line, "!STOP\n") == 0)       /* Kmon leaving */
          {
          orphan= 1;
          return;
          }
     if (strcmp(line, "!CONT\n") == 0)       /* Kmon is back */
          {
          orphan=0;
          paint();
          return;
          }
     stdin->_cnt= 0;          /* purge the stdin */
     n= sscanf(line, "%s %s %f", &keyword, &name, &data);
     if (n==0)
```

```
        {
        scprint(2, 20, 0, "trivial input line");
        goto end;
        }
    ptr= &keyword[0];
    switch(*ptr)
    {
    case 'f':
        flag();   /* control flag manipulator */
        break;
    case 'g':
        gman();   /* global variable manipulator */
        break;
    case 'i':
        init();   /* data acquisition initializer */
        break;
    case 'm':
        mon();    /* temporary, used to call monitor */
        break;
    case 'c':
        cleanup();     /* write current status of variables to file */
        break;
    case 'd':
        if (datflg)
            {
            scprint(2, 20, 0, "Data colection in progress.");
            break;
            }
        if (*name)
            {
            strcpy(dfname, name);
            }
        dfptr= fopen(dfname, "a");
        datflg= 1;     /* opens path in doalarm() for data acq. */
        scprint(2, 19, 0, "Data Sample Initiated.");
        scprint(27, 19, 0, "file= %s      ", dfname);
        moveto(99, 99);
        break;
    case 'p':
        paint();  /* re-paint the screen */
        break;
    case 'h':
        heat();        /* temporary used to initialize heater */
        break;
    case 'e':
        eng();         /* set api, ftmp, and point */
        break;
    default:
        scprint(2, 20, 0, "First character invalid in keyword.");
    }
end:
```

```
        *keyword= ' ';
        *name= ' ';
        data = 0;

        clear();
        moveto(99, 99);
}


/******                      GMAN()                      ******/
/*
gman is the funtion which manipulates global variables.
Input to the function is provided by keyboard().
Variables are either read or written based on the
strings: keyword[], name[], data. */

gman()
{
        char *ptr;      /* ptr to keyword string */
        char *nptr;     /* ptr to name string */
        struct gv *gvptr;   /* ptr to global variable names and addre */
        int match= 1;
        ptr= &keyword[2];
        nptr= name;
        gvptr= gvs;

        if (*name == 0)
                {
                scprint(2, 20, 0, "Must have a variahle name.");
                goto varerr;
                }

        while (match)
                {
                /* check if we ran through the list without a match */
                if (*gvptr->gvname == 0)
                        {
                        scprint(2, 20, 0, "Variable undefined.");
                        goto varerr;
                        }
                /* a match will yeild a '0' */
                match= strcmp(name, gvptr->gvname);
                gvptr++;
                }
        gvptr--;

        if (*ptr == 'r')
                scprint(2, 20, 0, "%s %f", gvptr->gvname, *(gvptr->gvad));
        else if (*ptr == 'w')
                {
```

```
          *(gvptr->gvad)= data;
          scprint(2, 20, 0, "%s %f", gvptr->gvname, *(gvptr->gvad));
          }
     else
          scprint(2, 20, 0, "Global variable must be read or write.");
varerr:
     ;
}



/******                    FLAG()                         ******/
/*
flag is the function which manipulates control flags.
Input to the function is provided by keyboard().
Flags are either read or written based on the
strings: keyword[], name[], data. */

flag()
{
     char *ptr;      /* ptr to keyword string */
     char *nptr;     /* ptr to name string */
     struct flg *flgptr; /* ptr to control flag names and addre */
     int match= 1;
     ptr= &keyword[2];
     nptr= name;
     flgptr= flags;

     if (*name == 0)
          {
          scprint(2, 20, 0, "Must have a flag name.");
          goto varerr;
          }
     while (match)
          {
          /* check if we ran through the list without a match */
          if (*flgptr->flgname == 0)
               {
               scprint(2, 20, 0, "Flag undefined.");
               goto varerr;
               }
          /* a match will yeild a '0' */
          match= strcmp(name, flgptr->flgname);
          flgptr++;
          }
     flgptr--;

     if (*ptr == 'r')
          scprint(2, 20, 0, "%s %d", flgptr->flgname, *(flgptr->flgad));
     else if (*ptr == 'w')
          {
```

```
          *(flgptr->flgad)= data;
          scprint(2, 20, 0, "%s %d", flgptr->flgname, *(flgptr->flgad));
          }
     else
          scprint(2, 20, 0, "Flag variablc must be read or write.");
varerr:
     ;
}


/******                        INIT()                         ******/
/* init routine.
     Purpose:  This routine is called by the keyboard servicc
     routine.  It interprets the second letter of the keyword
     when thc first letter was a 'i'.  The routines called
     by init are used to initialize the adac cards
     to the real world sensors.  */

init()
{
     char *ptr;     /*ptr to keyword */
     ptr= &keyword[1];
     switch(*ptr)
     {
     case 'h':
          inhl();   /* high level initialization routine */
          break;
     case 't':
          intmp();       /* temp initialization routinc */
          break;
     case 'p':
          intpul();      /* pulse card initialization */
          break;
     default:
          scprint(2, 20, 0, "Second letter invalid in kcyword.");
     }
}


/******                        INHL()                         ******/
/* inhl routine.
     Purpose:  High level data acquistion initialization
     routine.  By calling, the user can assign channel
     numbcrs, intcrnal gains, offsets, and calibration contsants
     to the fixed high level parameters.
     The routine uses the high level data acquistion structure
     and an option is available to print the status of
     the structure given the parameter namc. */
```

```
inhl()
{
     char *ptr;     /* ptr to keyword */
     struct hlda *hlptr; /* ptr to high level structure */
     int match= 1;
     hlptr= hldax;
     ptr= &keyword[2];

     if (*name == 0)
          {
          scprint(2, 20, 0, "Must have a parameter name.");
          goto parerr;
          }

     /* find the desired parameter in name */
     while (match)
          {
          if (*hlptr->parm == 0)
               {
               scprint(2, 20, 0, "High level parmeter undefined.");
               goto parerr;
               }
          match= strcmp(name, hlptr->parm);
          hlptr++;
          }
     hlptr--;
     /* find out what to do with this parameter */
     switch(*ptr)
     {
     case 'n':
          (hlptr->chan)= data;     /* assign channel */
          break;
     case 'g':
          (hlptr->gain)= data;     /* assign gain */
          break;
     case 'o':
          /* take 9 readings and assign mean to offset */
          (hlptr->off)= admean(hlptr->chan, hlptr->gain, 9);
          break;
     case 'c':
          /* take the mean of 9 readings and the data to form calcon */
          (hlptr->calcon)= data/((admean(hlptr->chan, hlptr->gain, 9))
               - hlptr->off);
          break;
     case 'p':
          /* print the status of the parameter */
          scprint(2, 20, 0, "%s %d %d %f %f", hlptr->parm, hlptr->chan,
               hlptr->gain, hlptr->off, hlptr->calcon);
          break;
     default:
          scprint(2, 20, 0, "Third letter in keyword not defined.");
```

```
    }
parerr:
    ;
}


/******                  INTMP()                  ******/
/* intmp routine.
    Purpose;  Temperature data acquistion initialization
    routine.  By calling, the user can assign channel
    numbers, and thermocouple types to the fixed
    temperature parameters.  The routine uses the
    temperature data acquisition structure
    and an option is available to print the status
    of the structure given the parameter name. */

intmp()
{
    char *ptr;      /* ptr to keyword */
    struct tmpda *tmptr;     /* ptr to temp structure */
    int match= 1;
    tmptr= tmpdax;
    ptr= &keyword[2];

    if (*name == 0)
        {
        scprint(2, 20, 0, "Must have a parameter name.");
        goto parerr:
        }

    /* find the desired parameter in name */
    while (match)
        {
        if (*tmptr->parm == 0)
            {
            scprint(2, 20, 0, "Temperature parameter undefined.");
            goto parerr;
            }
        match= strcmp(name, tmptr->parm);
        tmptr++;
        }
    tmptr--;
    /* find out what to do with this parameter */
    switch(*ptr)
    {
    case 'n':
        (tmptr->chan)= data;     /* assign channel */
        break;
    case 't':
        (tmptr->type)= data;     /* assign type */
        break;
```

```
     case 'p':
          /* print status of the parameter */
          scprint(2, 20, 0, "%s %d %d", tmptr->parm, tmptr->chan,
               tmptr->type);
          break;
     default:
          scprint(2, 20, 0, "Third letter in keyword not defined.");
     }
parerr:
     ;
}


/******                        INTPUL()                        ******/
/*
intpul routine.
     Purpose:  Pulsc card data acquisition initialization
     routine. By calling the user can assign channel numbers,
     time base, and the intercept and first order slope
     term to transform pulse data to real world units.
     The routine uses the pulse data acquistion structure
     and an optlon ls available to print the status
     of the structure given the parameter name. */

intpul()
{
     char *ptr;     /* ptr to keyword */
     struct pulse *pulptr;     /* ptr to pulse data structure */
     int match= 1;
     pulptr= pulscx;
     ptr= &keyword[2];

     if (*name == 0)
          {
          scprint(2, 20, 0, "Must have a paramctcr name.");
          goto parerr;
          }
     /* find the desired parameter in namc */
     while (match)
          {
          if (*pulptr->parm == 0)
               {
               scprint(2, 20, 0, "Pulse card parameter undefined.");
               goto parerr;
               }
          match= strcmp(name, pulptr->parm);
          pulptr++;
          }
     pulptr--;
     /* find out what to do with the parameter */
```

```
     switch(*ptr)
     {
     case 'n':
          (pulptr->chan)= data;     /* assign channel */
          break;
     case 't':
          (pulptr->tbase)= data;    /* assign time base */
          break;
     case 'o':
          (pulptr->const1)= data;  /* assign intercept */
          break;
     case 'c':
          (pulptr->const2)= data;  /* assign F.O. slope */
          break;
     case 'p':
          /* print status of the parameter */
          scprint(2, 20, 0, "%s %d %f %f %f", pulptr->parm,
               pulptr->chan, pulptr->tbase, pulptr->const1,
               pulptr->const2);
          break;
     default:
          scprint(2, 20, 0, "Third letter in keyword undefined.");
     }
parerr:
     ;
}


/******                        ENG()                            ******/
/*
Eng routine.
     This routine allows the user to set the fuel temp and API
     number.  In addition a label can be attached to the data
     collection sequence. */

eng()
{
     char *nptr;

     nptr = name;
     switch(*nptr)
          {
          case 'p':
               nptr++;
               strcpy(point,nptr);
               scprint(2, 20, 0, "point = %s", point);
               break;
          case 'a':
               if (data) (
                    api = data;
```

```
                        scprint(2, 20, 0, "api = %f", api);
                        }
                else {
                        scprint(2, 20, 0, "Data must have value");
                        }
                break;
        case 'f':
                if (data) {
                        ftmp = data;
                        scprint(2, 20, 0, "ftmp = %f", ftmp);
                        }
                else {
                        scprint(2, 20, 0, "Data must have value");
                        }
                break;
        default:
                scprint(2, 20, 0, "eng variable undefined");
    }
}


/******                    PAINT()                    ******/
/*
Paint Routine:
    This routine paints or re-paints the static screen layout. */

paint()
{
    scprint( 2,  2, 1, "ENGINE ");
    scprint( 2,  4, 0, "ERPM= ");
    scprint( 2,  5, 0, "ETRK= ");
    scprint( 2,  6, 0, "ENFF= ");
    scprint( 2,  7, 0, "EEGT= ");
    scprint( 2,  8, 0, "EWTP= ");
    scprint( 2,  9, 0, "ENOT= ");
    scprint( 2, 10, 0, "ENOP= ");
    scprint( 2, 11, 0, "THST= ");
    scprint( 2, 12, 0, "RACK= ");
    scprint( 2, 13, 0, "PNT = ");
    scprint( 2, 14, 0, "MT = ");
    scprint( 2, 15, 0, "FTMP= ");
    scprint(18,  2, 1, "CVT ");
    scprint(17,  4, 0, "TRPM= ");
    scprint(17,  5, 0, "TTRK= ");
    scprint(17,  6, 0, "TWI = ");
    scprint(17,  7, 0, "TOO = ");
    scprint(17,  8, 0, "TCVT= ");
    scprint(17,  9, 0, "TTNK= ");
    scprint(17, 10, 0, "TOI = ");
    scprint(17, 11, 0, "TWO = ");
```

```
     scprint(17, 12, 0, "OFF = ");
     scprint(17, 13, 0, "OFR = ");
     scprint(17, 14, 0, "RGST= ");
     scprint(17, 15, 0, "HSET= ");
     scprint(17, 16, 0, "FBST= ");
     scprint(34,  2, 1, "TRANS ");
     scprint(33,  4, 0, "PSOT= ");
     scprint(33,  5, 0, "FOOT= ");
     scprint(33,  6, 0, "GEAR= ");
     scprint(49,  2, 1, "DYNO ");
     scprint(49,  4, 0, "ORPM= ");
     scprint(49,  5, 0, "DTRK= ");
     scprint(49,  6, 0, "OWIT= ");
     scprint(49,  7, 0, "OWOT= ");
     scprint(33,  7, 0, "AIRT= ");
     scprint(66,  4, 0, "EPWR= ");
     scprint(66,  5, 0, "TPWR= ");
     scprint(66,  6, 0, "OPWR= ");
     scprint(66,  7, 0, "TEFF= ");
     scprint(33,  9, 1, "                                        ");
     scprint(46, 10, 1, "PROGRAM CONTROL ");
     scprint(33, 12, 0, "TIMINT= ");
     scprint(33, 13, 0, "MONTIM= ");
     scprint(33, 14, 0, "DYCTIM= ");
     scprint(33, 15, 0, "HRCTIM= ");
     scprint(33, 16, 0, "OATTIM= ");
     scprint(33, 17, 0, "OATMAX= ");
     scprint(46, 12, 0, "MONFLG= ");
     scprint(46, 13, 0, "OYCFLG= ");
     scprint(46, 14, 0, "HRCFLG= ");
     scprint(46, 15, 0, "OATFLG= ");
     scprint(46, 16, 0, "ONLINE= ");
     scprint(60, 12, 1, "HTCONO ");
     scprint(63, 13, 0, "TTL= ");
     scprint(63, 14, 0, "O/A= ");
     scprint(60, 15, 1, "DYCOND ");

     moveto(99, 99);
}


/******                   CLEANUP()                    ******/
/*
Cleanup Routine:
     This routine allows the user to write the entire contents
     of the common data structure to an external file for later
     examination.  */

cleanup()
{
     FILE *fp, *fopen();
```

```
    struct gv *gvptr;
    struct hlda *hlptr;
    struct tmpda *tmptr;
    struct pulse *pulptr;

    fp= fopen("Clean_up", "w");
    gvptr= gvs;
    hlptr= hldax;
    tmotr= tmpdax;
    pulptr= pulsex;

    fprintf(fp, "Global variables.\n");
    while(*gvptr->gvname)
         {
         fprintf(fp, "%s %f\n", gvptr->gvname,
                  *(gvptr->gvad));
         gvptr++;
         }
    fprintf(fp, "High level parameters.\n");
    while(*hlptr->parm)
         {
         fprintf(fp, "%s %d %d %f %f\n", hlptr->parm,
              hlptr->chan, hlptr->gain,
              hlptr->off, hlotr->calcon);
         hlptr++;
         }
    fprintf(fp, "Temperature parameters.\n");
    while(*tmptr->parm)
         {
         fprintf(fp, "%s %d %d\n", tmptr->parm, tmptr->chan,
                  tmptr->type);
         tmptr++;
         }
    fprintf(fp, "Pulse card parameters.\n");
    while(*pulptr->parm)
         {
         fprintf(fp, "%s %d %f %f %f\n", pulptr->parm,
              pulptr->chan, pulptr->tbase,
              pulptr->const1, pulptr->const2);
         pulptr++;
         }
    fclose(fp);
}


/******                         SCPRINT()                         ******/
/*
Scprint Routine:
    This routine is a modified version of the one which appears in
    kmon.  */
```

```
/*VARARGS4*/
scprint(x, y, att, fmt, args)
int  x, y, att;
char *fmt;
{
     if (orphan)    /* Kmon not here */
          {
          if (att)  /* monitor is in trouble */
               {
               printf(" 7 7");
               }
          return;
          }
     movea(x, y, att);
     _doprnt(fmt, &args, stdout);
     attrib(0);
}
```

```
*************************************************************************
*
*
*    C-SOURCE LISTING FOR ADAC 1000 COMPUTER
*         CVT -- ENGINE PROJECT -- SUPERVISOR
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT D. FUNK
*    DATE:           4/5/85
*    FILE:           doalarm.c
*
*************************************************************************

#include <stdio.h>
#include <math.h>
#include <adac.h>
#include "control.h"

/******                       DOALARM()                          ******/
/*
Doalarm Routine.
     Doalarm provides the control allocation algorithms for all
     real time loops.  This routine will be called any time that
     the UNIX system alarm goes off. */

doalarm()
{
     static dyfirst= 1;
     static htfirst= 1;
     static dtfirst= 1;
     static mnfirst= 1;
     static dyncnt=  1;
     static heatcnt= 1;
     static datcnt=  1;
     static moncnt=  1;

     alarmflag--;
     if (dyfirst && dycflg) {
          dyfirst= 0;
          dyn();
          }
     if (htfirst && hrcflg) {
          htfirst= 0;
          hcontrol(1);
          }
     if (dtfirst && datflg) {
          dtfirst= 0;
          dat();
          }
     if (mnfirst && monflg) {
```

```
            mnfirst= 0;
            mon();
            }
        if (((dyncnt*timint) == dyctim) && dycflg) {
            dyncnt= 0;
            dyn();
            }
        if (((heatcnt*timint) == hrctim) && hrcflg) {
            heatcnt= 0;
            hcontrol(0);
            }
        if (((datcnt*timint) == dattim) && datflg) {
            datcnt= 0;
            dat();
            }
        if (((moncnt*timint) == montim) && monflg) {
            moncnt= 0;
            mon();
            }
        if (dycflg == 0) {
            dyfirst= 1;
            dyncnt=  0;
            }
        if (hrcflg == 0) {
            htfirst= 1;
            heatcnt= 0;
            }
        if (datflg == 0) {
            dtfirst= 1;
            datcnt=  0;
            }
        if (monflg == 0) {
            mnfirst= 1;
            moncnt=  0;
            }

        dyncnt++;
        heatcnt++;
        datcnt++;
        moncnt++;
}


/******                        DYN()                           ******/
/*
Dyn Routine.
     The dyn routine provides the real time control for the
     dynamometer.  At present this routine has not been completely
     developed. */
```

```
dyn()
{
     static int cnt= 0;

/* UNDEVELOPED */
     if (cnt == 0) {
          cnt = 1;
          scprint(2, 17, 0, "dyno loop *");
          goto end;
          }

     if (cnt == 1) {
          cnt = 0;
          scprint(2, 17, 0, "dyno loop* *");
          }
end:
     ;
}


/******                    HCONTROL()                    ******/
/*
Hcontrol Routine.
     The hcontrol routine provides the real time control
     for the cvt oil heater. This routine is fully developed. */

hcontrol(reset)
int reset;
{
     int out;
     static double told;
     double too, tcvt, ttnk, toi, ttap, control;
     double tmpmean();
     struct tmpda *tmptr;

     tmptr= &tmpdax[3];
     tcvt= tmpmean(tmptr->chan, tmptr->type, 3);
     tmptr= &tmpdax[8];
     too= tmpmean(tmptr->chan, tmptr->type, 3);
     tmptr= &tmpdax[10];
     ttnk= tmpmean(tmptr->chan, tmptr->type,3);
     tmptr= &tmpdax[11];
     toi= tmpmean(tmptr->chan, tmptr->type, 3);
     tmptr= &tmpdax[12];
     ttap= tmpmean(tmptr->chan, tmptr->type, 3);

     if (reset) {
          told= too;
          }
```

```
     control= 619.976*hset -210.098*too +467.982*told
              -749.86*(toi-ttnk) - 719.866*tcvt -125.0*ttap;
     told= too;

     scprint(67, 12, 0, "%8.2f", control);
     moveto(99,99);

     out= ceil(control);
     if (out <= 0) {
          out= 0;
          }
     if (out >= 4095) {
          out= 4095;
          }
     if (out < 2048) {
          ttlval= 0;
          daval= out;
          }
     if (out >= 2048) {
          ttlval= 1;
          daval= out -2048;
          }

     heat();
}


/******                       HEAT()                        ******/
/*
Heat Routine.
     Heat sets the outputs of the heaters. */

heat()
{
     dtoa(0, daval);
     ttlwb(2, ttlval);
}
```

```
***********************************************************************
*
*
*    C-SOURCE LISTING FOR ADAC 1000 COMPUTER
*         CVT -- ENGINE PROJECT -- SUPERVISOR
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT D. FUNK
*    DATE:          4/5/85
*    FILE:          mon.c
*
***********************************************************************

#include <stdio.h>
#include <adac.h>
#include "control.h"

/******                       MON()                              ******/
/*
Mon Rountine:
     Mon() is one of the real time loops.  It collects all necessary
     data and analyzes it.  Then it prints all data along with
     warnings to the static screen. */

mon()
{
     int  t[2];
     char *p;
     double tempf();
     double tmpmean();
     struct hlda *hlptr;
     struct tmpda *tmptr;
     struct pulse *pulptr;
     struct limits *hllptr;
     struct limits *tmplptr;
     struct limits *pullptr;
     int i, dat, att, err;
     double value, oiltemp, flow;
     double epwr, tpwr, dpwr, teff, mt;

     scprint(1, 21, 0, "                                                    ");

/* make request for data */
     err= do68k(2, 1);

     hlptr= hldax;
     tmptr= tmpdax;
     pulptr= pulsex;
     hllptr= hl_lim;
     tmplptr= tmp_lim;
```

```
    pullptr= pul_lim;


/* the following checks erpm, etrk, trpm, ttrk, drpm, dtrk */
    att= NORM;
    time(t);
    p= ctime(t);
    scprint(50, 1, 0, "%s", p);
    for (i=0; i<6; i++)
          {
          dat= atodi(hlptr->chan, hlptr->gain);
          value= (dat - hlptr->off)*(hlptr->calcon);
          if (value >= *(hllptr->warn))
                {
                if (value >= *(hllptr->max))
                      att= REVERSE;
                else
                      att= UNDERLINE;
                }
          scprint(hlptr->a, hlptr->b, att, "%8.1f", value);
          att= NORM;
          hlptr++;
          hllptr++;
          }

/* check engine oil pressure */
          dat= atodi(hlptr->chan, hlptr->gain);
          value= (dat - hlptr->off)*(hlptr->calcon);
          if (value <= *(hllptr->warn))
                {
                if (value <= *(hllptr->max))
                      att= REVERSE;
                else
                      att= UNDERLINE;
                }
          scprint(hlptr->a, hlptr->b, att, "%8.1f", value);
          att=NORM;
          hlptr++;


/* check fuel bucket (enff) */
          dat= atodi(hlptr->chan, hlptr->gain);
          value= (dat - hlptr->off)*(hlptr->calcon);
          if (value <= enffmn) {
                att= REVERSE;
                }

          scprint(hlptr->a, hlptr->b, att, "%8.1f", value);
          att= NORM;
```

```
/* check ewtp, eegt, enot, tcvt, twi, psot, fdot, dwot */
     for (i=0; i<8; i++)
          {
          value= tempf(tmptr->chan, tmptr->type);
          if (value >= *(tmplptr->warn))
               {
               if (value >= *(tmplptr->max))
                    att= REVERSE;
               else
                    att= UNDERLINE;
               }
          scprint(tmptr->c, tmptr->d, att, "%8.1f", value);
          att= NORM;
          tmptr++;
          tmplptr++;
          }

/* check cvt oil in (too) */
     value= tempf(tmptr->chan, tmptr->type);
     oiltemp= 67.8 -.63*value;
     if ((value-hset) >= *(tmplptr->max))
          att= REVERSE;
     if ((hset-value) >= *(tmplptr->warn))
          att= UNDERLINE;
     scprint(tmptr->c, tmptr->d, att, "%8.1f", value);
     att= NORM;

/* find MT for cvt temp map */
     tmptr= &tmpdax[8];
     value= tmpmean(tmptr->chan, tmptr->type, 5);
     tmptr= &tmpdax[3];
     mt= tmpmean(tmptr->chan, tmptr->type, 5);
     mt= (mt+value)/2.0;
     scprint(8, 14, att, "%8.1f", mt);


/* check ambient air temp */
     tmptr= &tmpdax[9];
     tmplptr++;
     value= tmpmean(tmptr->chan, tmptr->type, 5);
     1f (value >= *(tmplptr->max))
          att= REVERSE;
     if (value<= *(tmplptr->warn))
          att= UNDERLINE;
     scprint(tmptr->c, tmptr->d, att, "%8,1f", value);
     att= NORM;


/* check cvt oll flows */
     for(i=0; i<2; i++)
          {
```

```
            value= (pul(pulptr->chan))/(pulptr->lbase);
            flow= value/((pulptr->const1)*(pow(value/oiltemp, 0.5))
                           + (pulptr->const2));
            if (flow >= *(pullptr->max))
                att= REVERSE;
            if (flow<= *(pullptr->warn))
                att= UNDERLINE;
            scprint(pulptr->e, pulptr->f, att, "%8.1f", flow);
            att= NORM;
            pulptr++;
            pullptr++;
            )

/* update dwit, tol, ttnk */
        tmptr= &tmpdax[12];
        value= tempf(tmptr->chan, tmptr->type);
        scprint(tmptr->c, tmptr->d, 0, "%8.1f", value);

        tmptr= &tmpdax[11];
        value= tempf(tmptr->chan, tmptr->type);
        scprint(tmptr->c, tmptr->d, 0, "%8.1f", value);

        tmptr= &tmpdax[10];
        value= tempf(tmptr->chan, tmptr->type);
        scprint(tmptr->c, tmptr->d, 0, "%8.1f", value);

        tmptr= &tmpdax[13];
        value= tempf(tmptr->chan, tmptr->type);
        scprint(tmptr->c, tmptr->d, 0, "%8.1f", value);


/* compute power and efficiency stats */
        hlptr= &hldax[0];
        dat= admean(hlptr->chan, hlptr->gain, 5);
        epwr= (dat -hlptr->off)*(hlptr->calcon);
        hlptr++;
        dat= admean(hlptr->chan, hlptr->gain, 5);
        value= (dat -hlptr->off)*(hlptr->calcon);
        epwr= epwr*value/5252.1131;

        hlptr++;
        dat= admean(hlptr->chan, hlptr->gain, 5);
        tpwr= (dat -hlptr->off)*(hlptr->calcon);
        hlptr++;
        dat= admean(hlptr->chan, hlptr->gain, 5);
        value= (dat -hlptr->off)*(hlptr->calcon);
        tpwr= tpwr*value/5252.1131;

        hlptr++;
        dat= admean(hlptr->chan, hlptr->gain, 5);
        dpwr= (dat -hlptr->off)*(hlptr->calcon);
```

```
hlptr++;
dat= admean(hlptr->chan, hlptr->gain, 5);
value= (dat -hlptr->off)*(hlptr->calcon);
dowr= dpwr*value/4000.00;

scprint(72,4,0, "%6.1f", epwr);
scprint(72,5,0, "%6.1f", tpwr);
scprint(72,6,0, "%6.1f", dpwr);

if (epwr > 1) {
     teff= tpwr/epwr*100.0;
     att= NORM;
     if (teff < 75 ) {
          att = REVERSE;
          }
     scprint(72,7,att, "%6.1f", teff);
     }


/* update 68k table */
     err= do68k(2, 2);
     if (err == 3)
          scprint(2, 21, 0, "68000 not responding");
     if (err == 4)
          scprint(2, 21, 0, "wrong number of arguements in 68000");
     if (err == 5)
          scprint(2, 21, 0, "S flag missing in 68000");

     scprint( 8, 12, 0, "%8d", rack);
     scprint( 8, 11, 0, "%8d", thstp);
     scprint(23, 14, 0, "%8d", rgstp);
     scprint(23, 16, 0, "%8d", fbstp);
     scprint(39,  6, 0, "%8d", gear);


/* update the program control flags */
     scprint(40, 12, 0, "%4d", timint);
     scprint(40, 13, 0, "%4d", montim);
     scprint(40, 14, 0, "%4d", dyctim);
     scprint(40, 15, 0, "%4d", hrctim);
     scprint(40, 16, 0, "%4d", dattim);
     scprint(40, 17, 0, "%4d", datmax);
     scprint(53, 12, 0, "%2d", monflg);
     scprint(53, 13, 0, "%2d", dycflg);
     scprint(53, 14, 0, "%2d", hrcflg);
     scprint(53, 15, 0, "%2d", datflg);
     scprint(53, 16, 0, "%2d", online);
     scprint(67, 13, 0, "%6d", ttlval);
     scprint(67, 14, 0, "%6d", daval);
```

```
/* update heater control setpoint */
     scprint(23, 15, 0, "%8.1f", hset);
     scprint( 8, 13, 0, "%s   ", point);
     scprint( 8, 14, 0, "%8.1f", api);
     scprint( 8, 15, 0, "%8.1f", ftmp);

     moveto(99,99);
}
```

```
*************************************************************************
*
*
*    C-SOURCE LISTING FOR ADAC 1000 COMPUTER
*         CVT -- ENGINE PROJECT -- SUPERVISOR
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT D. FUNK
*    OATE:           4/5/85
*    FILE:           cvtlinear.c
*
*************************************************************************

#include <stdio.h>
#include <adac.h>
#include "control.h"

/******                      DAT()                              ******/
/*
Dat Routine.
     The dat routine provides the data rccording proceedures.
     There are numerous versions of this routine used for
     different tests.  This particular version was used to collect
     the CVT data. */

dat()
{
     int  t[2];
     char *p;
     struct hlda *hlptr;
     struct tmpda *tmptr;
     double tmpmean();
     int dat, err, errcnt, num, wait;
     static cnt=0;
     double enot, atmp, enop, cnwt, eegt, erpm, etrk, enff;
     double trpm, ttrk, tcvt, too;

     scprint(1,22,0,"                                                  ");

     num = cnt + 1;

     if (cnt == 0) {
          time(t);
          p= ctime(t);
          fprintf(dfptr, "%s", p);
          fprintf(dfptr, "%s %10.3f %10.3f\n", point, api, ftmp);

          tmptr= &tmpdax[0];
          enwt= tmpmean(tmptr->chan, tmptr->type, 9);
          tmptr++;
```

```
      eegt= tmpmean(tmptr->chan, tmptr->type, 9);
      tmptr++;
      enot= tmpmean(tmptr->chan, tmptr->type, 9);
      tmptr++;
      tcvt= tmpmean(tmptr->chan, tmptr->type, 9);
      tmptr= &tmpdax[8];
      too= tmpmean(tmptr->chan, tmptr->type, 9);
      tmptr= &tmpdax[9];
      atmp= tmpmean(tmptr->chan, tmptr->type, 9);
      hlptr= &hldax[6];
      dat= admean(hlptr->chan, hlptr->gain, 9);
      enop= (dat - hlptr->off) * hlptr->calcon;

fprintf(dfptr, "%9.3f %9.3f %9.3f %9.3f %9.3f %9.3f %9.3f\n",
                 enot, enop, enwt, eegt, tcvt, too, atmp);

      req= 1;
      scprint(50, 19, 0, "DATCNT= ");
      }

scprint(58, 19, 0, "%5d", num);
err= do68k(3, 1);

hlptr= &hldax[0];
dat= admean(hlptr->chan, hlptr->gain, 9);
erpm= (dat - hlptr->off) * hlptr->calcon;
hlptr++;
dat= admean(hlptr->chan, hlptr->gain, 9);
etrk= (dat - hlptr->off) * hlptr->calcon;
hlptr++;
dat= admean(hlptr->chan, hlptr->gain, 9);
trpm= (dat - hlptr->off) * hlptr->calcon;
hlptr++;
dat= admean(hlptr->chan, hlptr->gain, 9);
ttrk= (dat - hlptr->off) * hlptr->calcon;
hlptr= &hldax[7];
dat= admean(hlptr->chan, hlptr->gain, 9);
enff= (dat - hlptr->off) * hlptr->calcon;

errcnt= 0;
while (errcnt < 1000) {
          errcnt++;
          wait= 0;
          while (wait < waitmx) {
               wait++;
               }
          }
err= do68k(3, 2);
if (err == 1) {
     fprintf(dfptr, "**Error 68k Not Here\n");
     }
```

```
    if (err == 3) {
         fprintf(dfptr, "**Error 68k Not Responding\n");
         scprint(2, 22, 0, "Error 68k Not Responding");
         }
    if (err == 4) {
         fprintf(dfptr, "**Error 68k Argument Error\n");
         scprint(2, 22, 0, "Error 68k Argument Error");
         }
    if (err == 5) {
         fprintf(dfptr, "**Error 68k Format Error\n");
         scprint(2, 22, 0, "Error 68k Format Error");
         }

    fprintf(dfptr, "%9.3f %9.3f %9.3f %8d %8d %8d %9.3f %9.3f\n",
              erpm, etrk, enff, thstp, rack, fbstp, trpm, ttrk);

    cnt++;
    if (cnt >= datmax)
         {
         datflg= 0;
         cnt= 0;
         req= 0;
         fprintf(dfptr, "\n");
         fclose(dfptr);
         scprint( 2, 19, 0, "                         ");
         scprint(50, 19, 0, "                         ");
    scprint(1,22,0,"                                               ");
    }

    moveto(99,99);
}
```

APPENDIX C

MC68000ECB HARDWARE

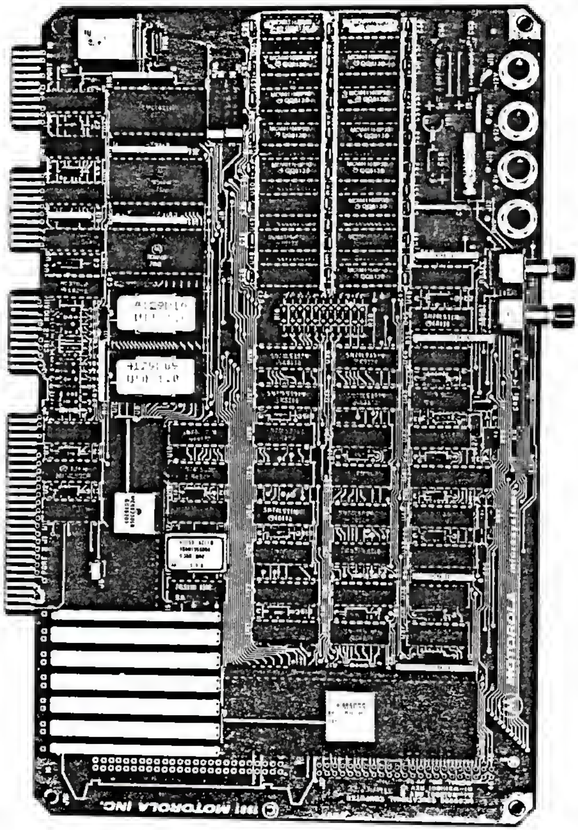


Figure 7 MC68000 Educational Computer Board

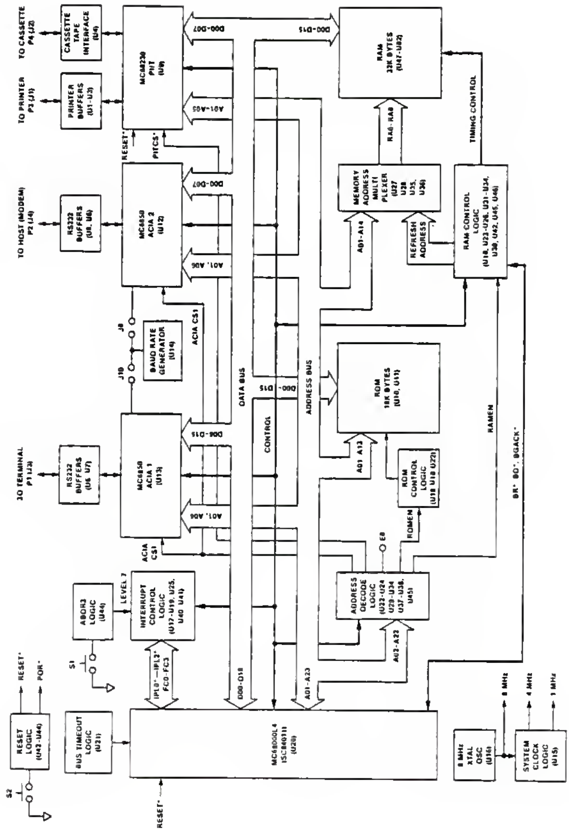


Figure 8 Block Diagram - Educational Computer Board

Table 3 Bus Expansion Buffer: Parts List

REFERENCE DESIGNATION	DESCRIPTION	
R1, R2, R4	Resistor, film, 270 ohm, 5%, 1/4 W	
R3	Resistor, film, 180 ohm, 5%, 1/4 W	
R5, R6	Resistor, film, 3.3k ohm, 5%, 1/4 W	
U1	I.C. SN74LS245N	Octal Bus Transceiver
U2, U3, U4	I.C. SN74LS244N	Octal Bus Buffer
U5	I.C. SN74LS07N	Hex Buffer
U6	I.C. SN74LS11N	Triple Three AND Gates
U7	I.C. SN74LS04N	Hex Inverter

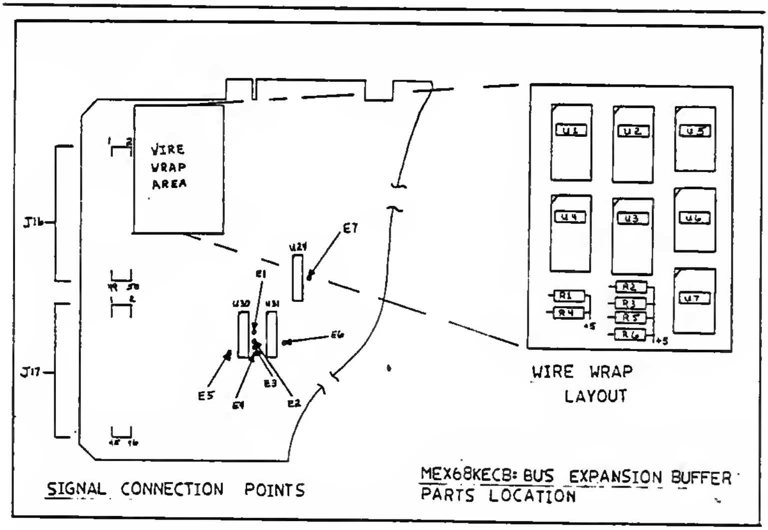


Figure 9 Bus Expansion Buffer: Board Layout

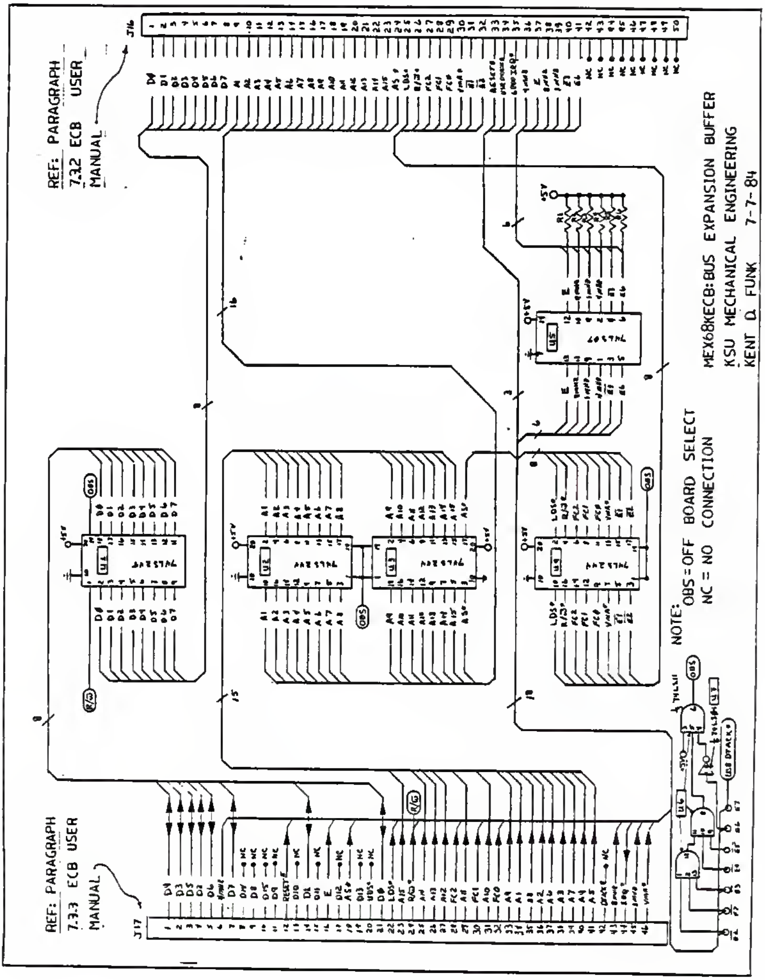


Figure 10 Bus Expansion Buffer: Schematic

Table 4 I/O Expansion: Parts List

REFERENCE DESIGNATION	DESCRIPTION	
R1, R2	Resistor, film, 68k ohm, 5%, 1/4 W	
R3, R5, R7, R9, R11, R13, R15, R17, R29, R32, R33, R35, R37, R39, R41, R43, R51, R53, R55, R57, R59, R61, R63, R65	Resistor, film, 220 ohm, 5%, 1/4 W	
R4, R6, R8, R10, R12, R14, R16, R18, R30, R31, R34, R36, R38, R40, R42, R44, R52, R54, R56, R58, R60, R62, R64, R66	Resistor, film, 390 ohm, 5%, 1/4 W	
R19, R20, R67, R68, R69, R70, R71, R72, R73, R74	Resistor, film, 4.7k ohm, 5%, 1/4 W	
R21, R22, R23, R24, R25, R26, R27, R28	Resistor, film, 1k ohm, 5%, 1/4 W	
R45, R46, R47, R48, R49, R50	Resistor, film, 2.2k ohm, 5%, 1/4 W	
U1	I.C. SN74LS260N	Dual 5-input NOR
U2	I.C. SN74LS138N	3-to-8 Line Decoder
U3, U4	I.C. MC68230L8	PI/T
U5, U6, U7, U8, U9	I.C. SN74LS90N	Decade Counter

Table 4 --cont.

REFERENCE DESIGNATION	DESCRIPTION	
U10	I.C. SN74LS08N	Quad 2-Input AND
U11, U12	I.C. SAA1042	Stepper Motor Driver
U13, U14, U25, U32, U33	I.C. SN74LS05N	OC Hex Inverter
U15, U16, U17, U18, U19, U20, U21, U22, U23, U24, U26, U27, U28, U29, U30, U31, U34, U35, U36, U37, U38, U39, U40, U41	I.C. 4N33	Opto-Isolator
U42, U43	Spaulding's Incremental Encoder Counters	

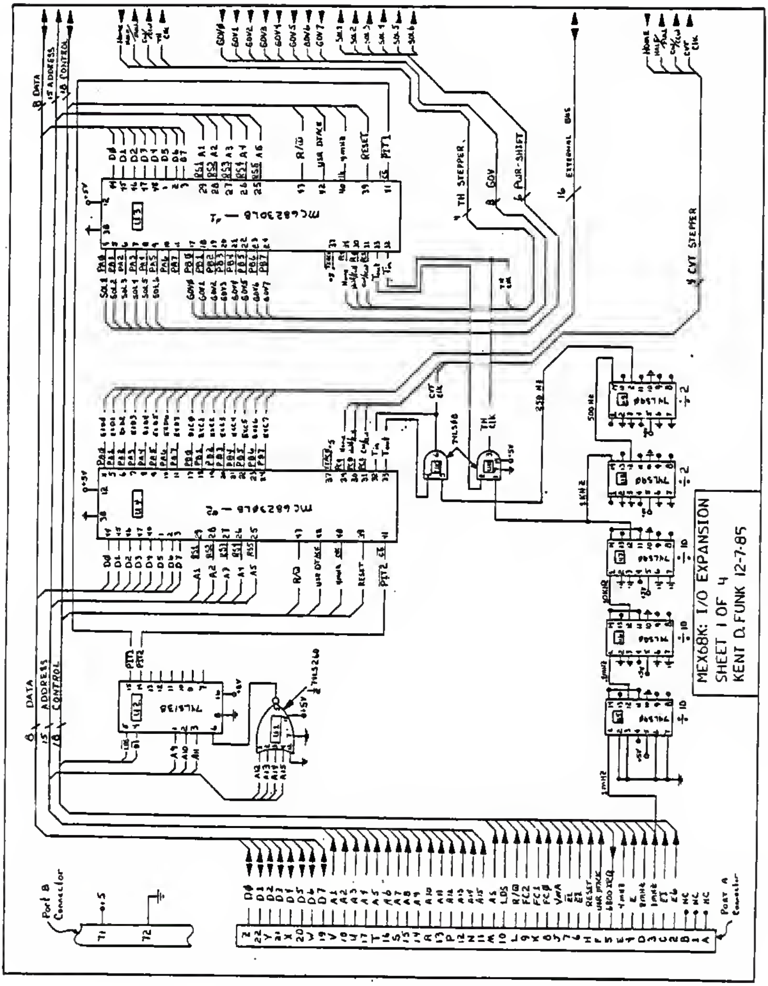


Figure 11 I/O Expansion: Schematic

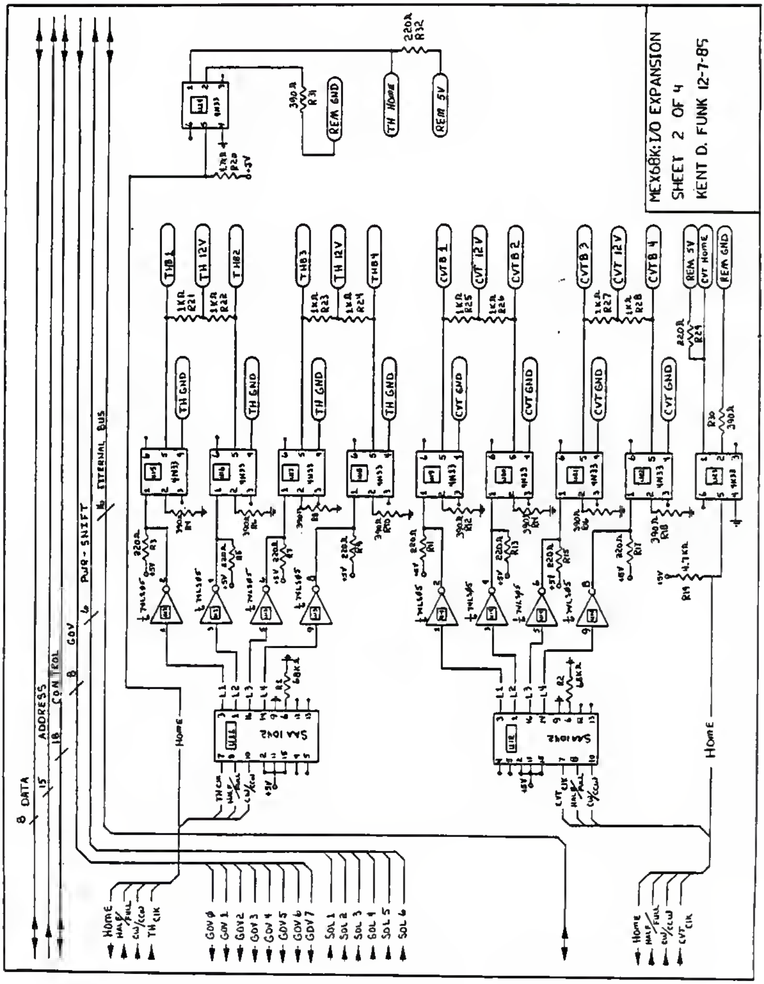


Figure 11 --cont.

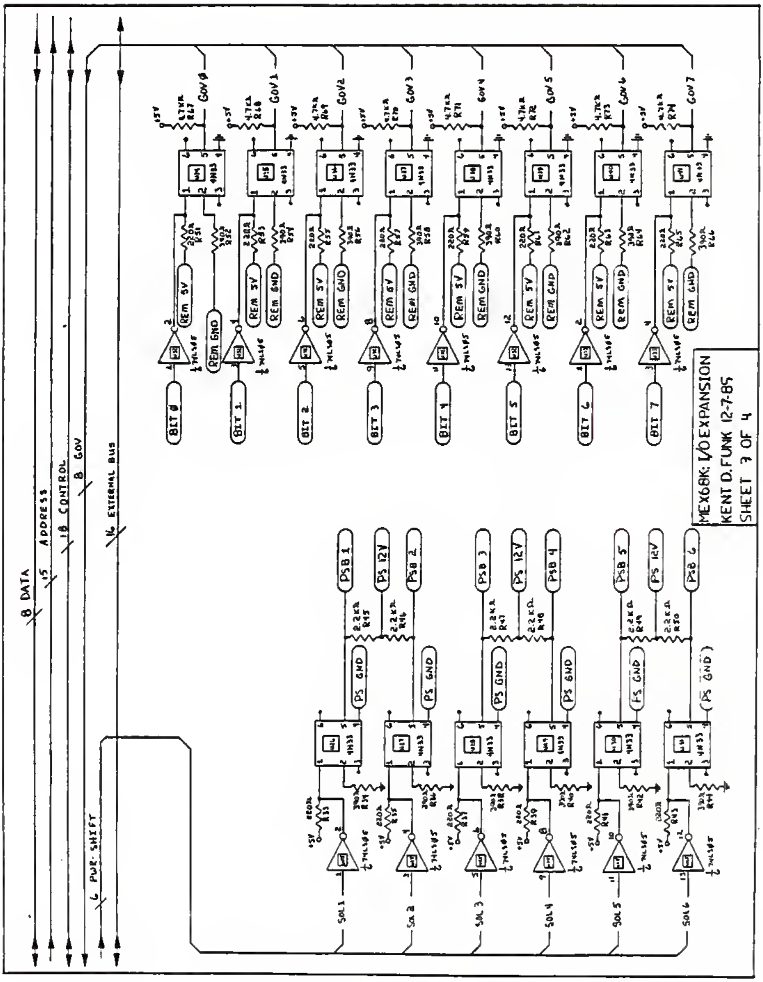


Figure 11 --cont.

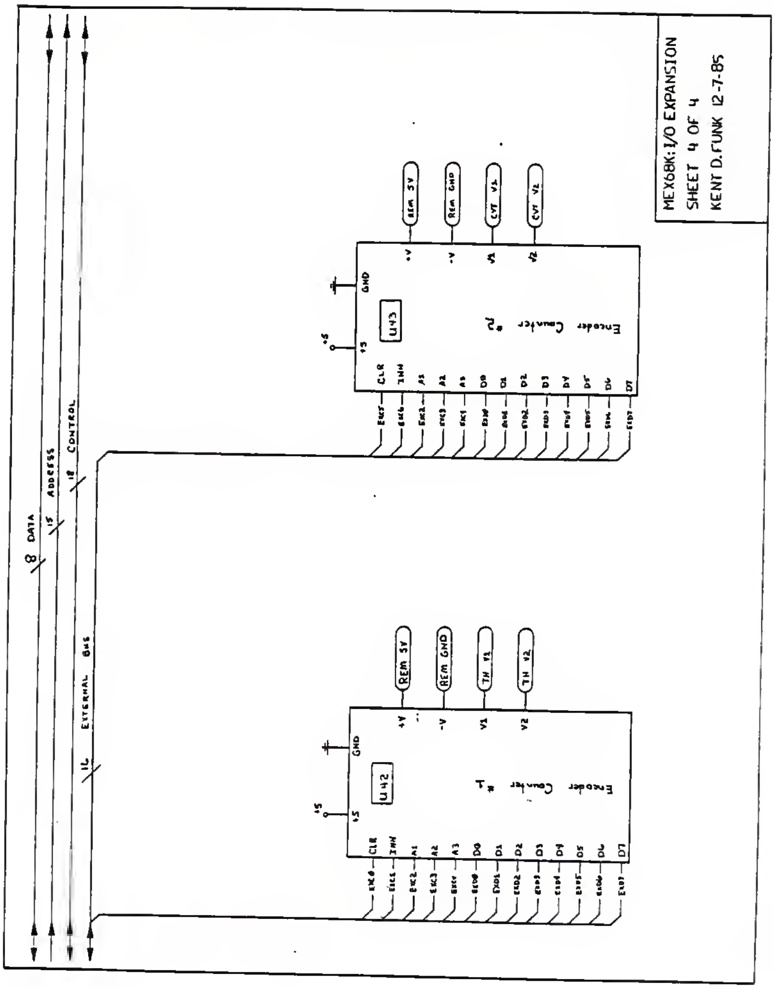


Figure 11 --cont.

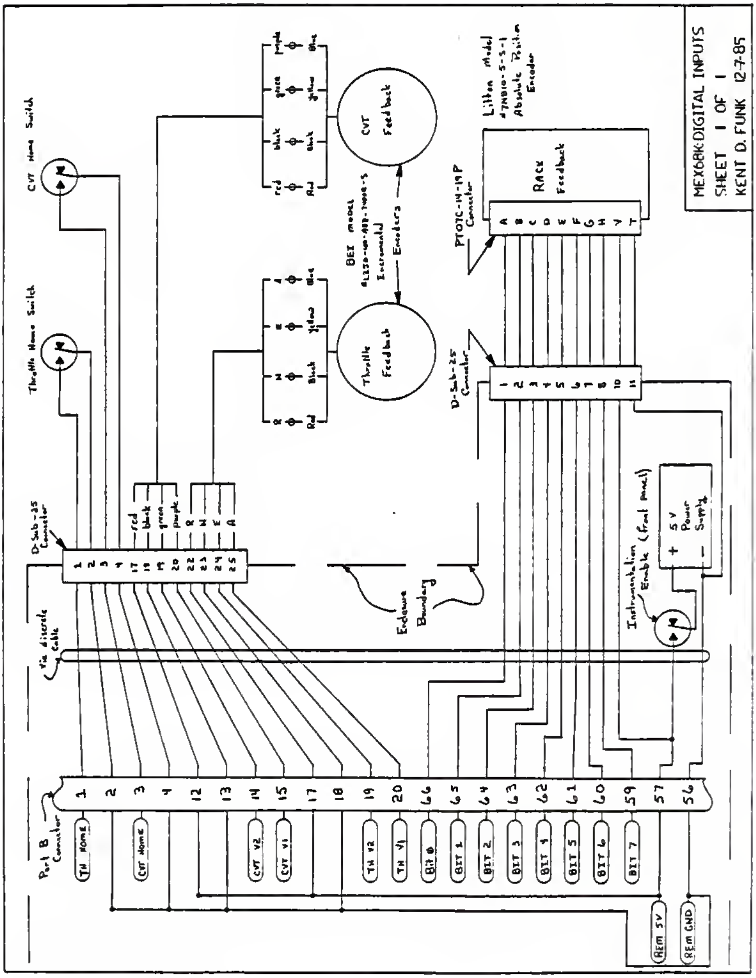


Figure 12 Digital Inputs: Schematic

Table 5 High Current Outputs: Parts List

REFERENCE DESIGNATION	DESCRIPTION	
D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14	SK5040	Silicon Fast Recovery Rectifier
R1, R2	Resistor, power, 2.6 ohm, 10%, 20 W	
R3, R4	Resistor, power, 1.1 ohm, 10%, 65 W	
T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14	SK3180	NPN Si AF Darlington Transistor

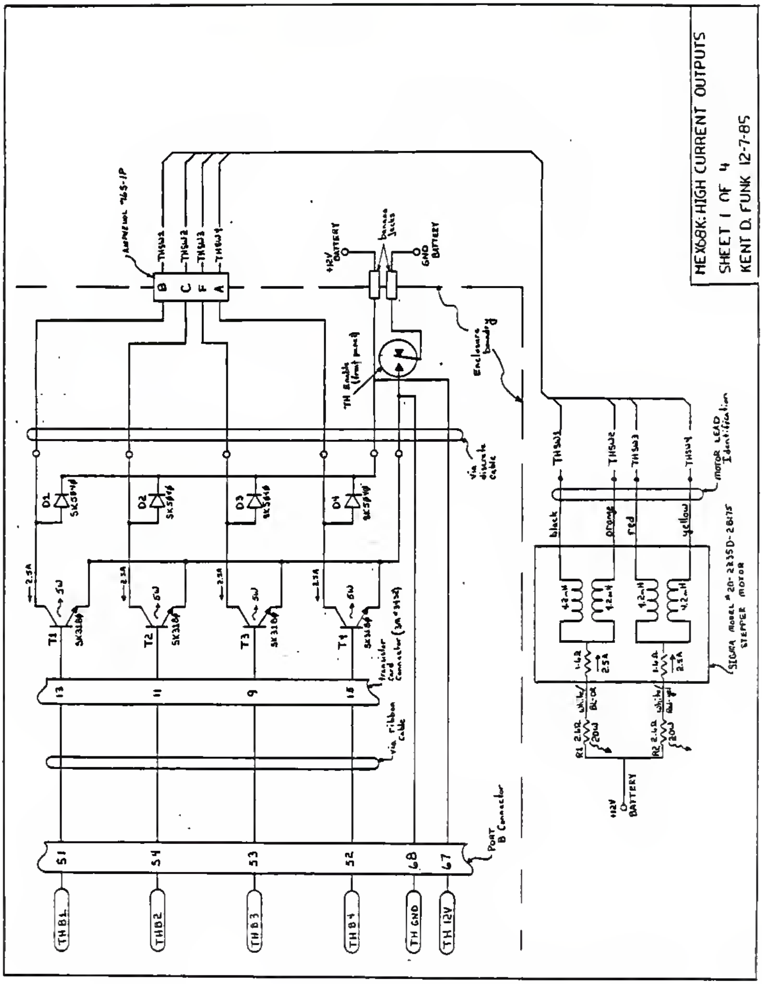


Figure 13 High Current Outputs: Schematic

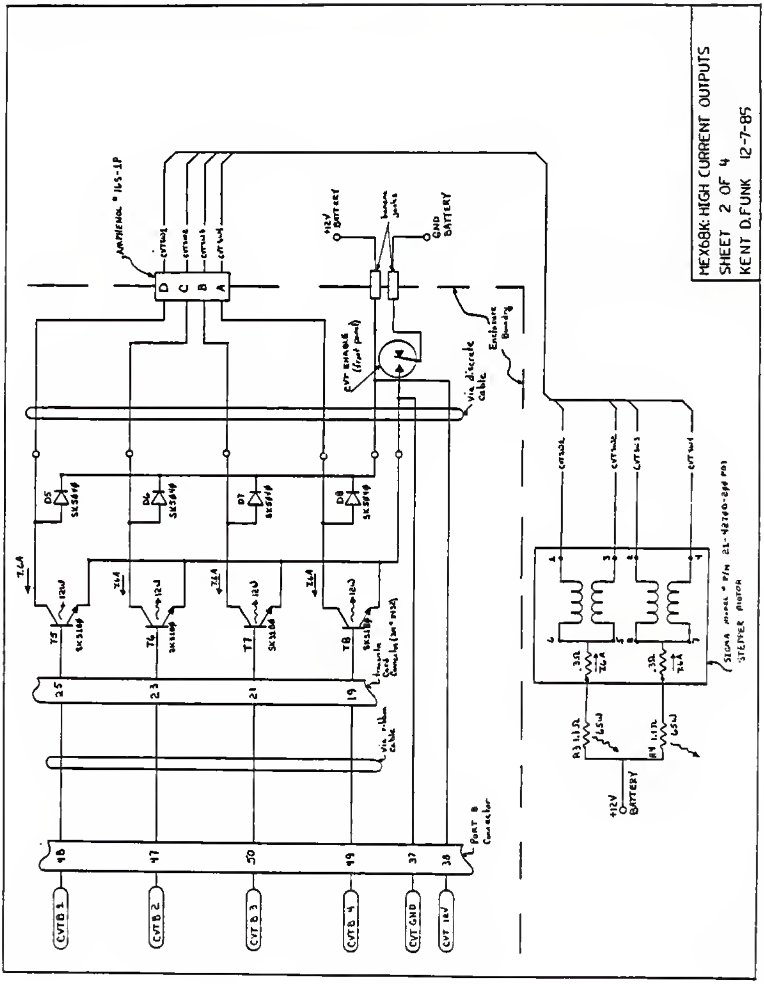


Figure 13 --cont.

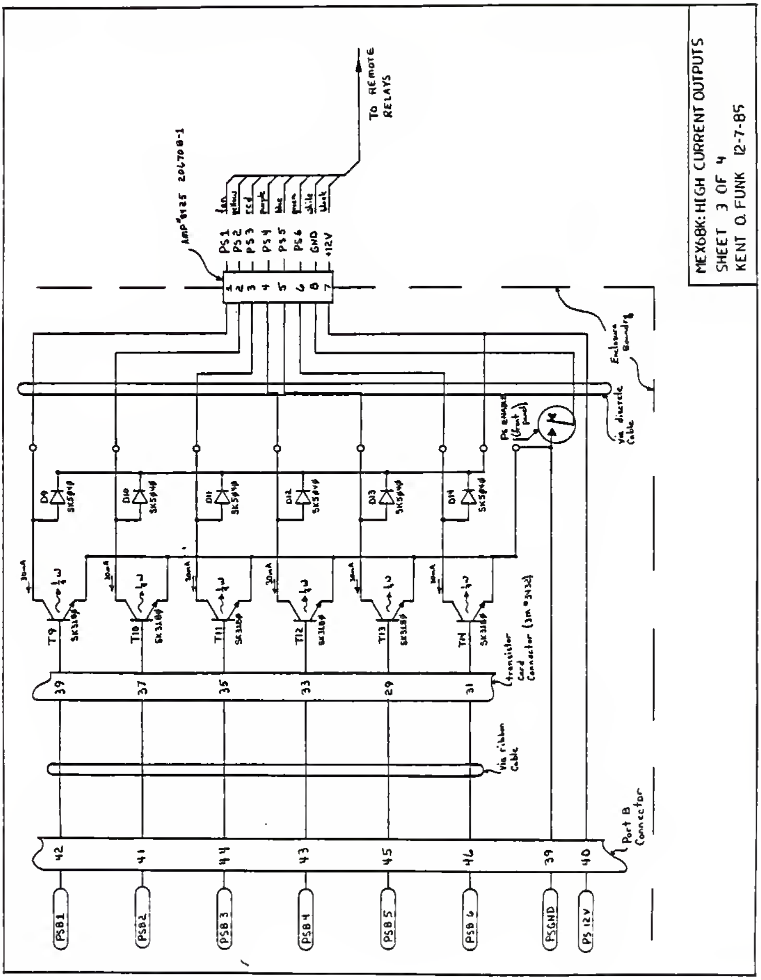


Figure 13 --cont.

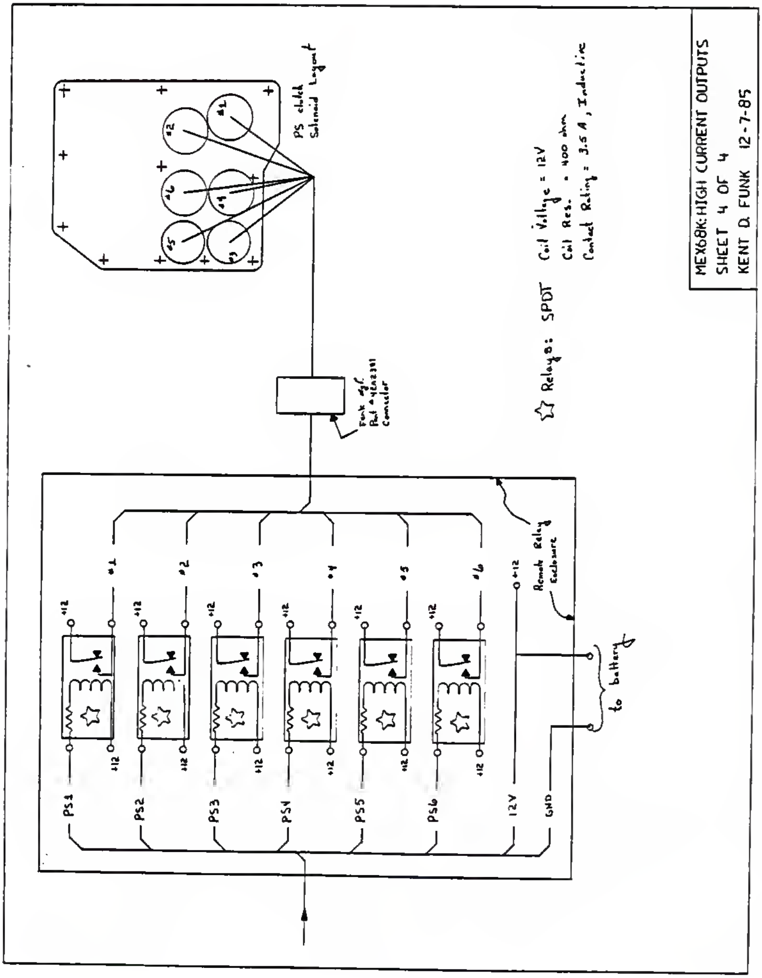


Figure 13 --cont.

APPENDIX D

MC68000ECB SOFTWARE LISTINGS

```
**********************************************************************
*
*
*    SUPPLEMENTAL LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    OEPARTMENTS OF MECHANICAL ANO AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT O. FUNK
*    DATE:          8/17/85
*    FILE:          screenlayout
*
**********************************************************************


          *CONSOLE SCREEN LAYOUT*

P-INIT
E-INIT
    SET ENABLES ON
        CONTINUE?  <ret>
    START ENGINE
        CONTINUE?  <ret>

CVT CONTROLLER
    CREATEO BY K FUNK


CURRENT STATUS:
RACK=   XXXXXXXX
THSTP=  XXXXXXXX
RGSTP=  XXXXXXXX
FBSTP=  XXXXXXXX
GEAR=   XXXXXXXX

ENTER NEW OATA:
THSTP ? <user response>
RGSTP ? <user response>
GEAR ?  <user response>
```

```
***********************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT D. FUNK
*    DATE:          B/17/85
*    FILE:          global.s
*    TYPE:          Common Data Structures -- global
*
***********************************************************************

DATA STRUCTURES:

     GLOBAL PROGRAM SCOPE...
          global program data allocation ($900 to $DFF)

     -------------------------------------------------------------------
     000900     .
     .          .
     .    console buffer (80 ASCII characters)
     .          .
     .          .
     0009A0-------- (Console buffer End)
     -------------------------------------------------------------------

     -------------------------------------------------------------------
     0009B0  hbyte   RACK          (current external conditions)
     0009B1  lbyte

     0009B2  hbyte  THSTP
     0009B3  lbyte

     0009B4  hbyte  RGSTP
     0009B5  lbyte

     0009B6  hbyte  FBSTEP
     0009B7  lbyte

     0009B8  hbyte  GEAR
     0009B9  lbyte
     -------------------------------------------------------------------

     -------------------------------------------------------------------
     0009C0  hbyte  THSTP          (desired external conditions)
     0009C1  lbyte

     0009C2  hbyte  RGSTP
```

```
0009C3  lbyte

0009C4  hbyte  GEAR
0009C5  lbyte
-----------------------------------------------------------------
```

GLOBAL TRAP #14 SCOPE...
global trap #14 data allocation ($5000 to $507F)

```
-----------------------------------------------------------------
005000= $00006000   (p-init)       (trap extension table)
005004= $01006100   (e-init)
005008= $02006200   (move-th)
00500C= $03006300   (move-cvt)
005010= $04006400   (set-gear)
005014= $05006500   (read-rack)
005018= $06006600   (test-stable)
00501C= $07006700   (reg-save)
005020= $08006800   (reg-restore)
005024= $09006900   (read-thstp)
005028= $0A006A00   (setup)
00502C= $0B006950   (read-fbstp)
005030= $0C006D00   (unassigned)
005034= $0D006E00   (unassigned)
005038= $0E006F00   (unassigned)
00503C= $0F006F80   (unassigned)
005040= $00000000   (reserved for link address to ROM table)
005044--------      (extension table End)
-----------------------------------------------------------------
```

```
*************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT D. FUNK
*    DATE:           8/17/85
*    FILE:           main.s
*    TYPE:           Sequential Program Execution -- main
*
*************************************************************************
```

SYNOPSIS:

MAIN provides the control code for all normal sequential program execution. Throughout the main program, numerous software interrupts are generated (TRAP #14) in order to initialize and communicate with all system hardware. Currently, the main LOOP provides an interactive environment (not real time controlled) by which the user can manipulate the drive train. This configuration is useful for data mapping and system debugging.

To Start the Program;

GO 1000

OATA STRUCTURES:

GLOBAL STRUCTURES AFFECTED:

PROGRAM--
- current external conditions
- desired external conditions

TRAP #14--
- trap extension table

LOCAL MAIN SCOPE...

local data allocation ($E00 to $FFF)

```
000E00= $0D0A            (Header String)
000E02= 'CV'
000E04= 'T '
000E06= 'CO'
000E08= 'NT'
000E0A= 'RO'
000E0C= 'LL'
```

```
000E0E= 'ER'
000E10= $0D0A
000E12= $2020
000E14= $2020
000E16= 'CR'
000E18= 'EA'
000E1A= 'TE'
000E1C= 'D '
000E1E= 'BY'
000E20= ' K'
000E22= ' F'
000E24= 'UN'
000E26= 'K '
000E28= $0D0A
000E2A= $0A0A
000E2C= 'CU'
000E2E= 'RR'
000E30= 'EN'
000E32= 'T '
000E34= 'ST'
000E36= 'AT'
000E38= 'US'
000E3A= ': '
000E3C= $0D0A
000E3E-------- (Header String End)
--------------------------------------------------------------------

--------------------------------------------------------------------
000E40= 'RA'              (Data String)
000E42= 'CK'
000E44= '= '
000E46= $2020  (Rack-Start= $E46)
000E48= $2020
000E4A= $2020
000E4C= $2020
000E4E= $000A

000E50= 'TH'
000E52= 'ST'
000E54= 'P='
000E56= $2020  (TH-Start= $E56)
000E58= $2020
000E5A= $2020
000E5C= $2020
000E5E= $0D0A

000E60= 'RG'
000E62= 'ST'
000E64= 'P='
000E66= $2020  (RG-Start= $E66)
000E68= $2020
```

```
000E6A= $2020
000E6C= $2020
000E6E= $0D0A

000E70= 'FB'
000E72= 'ST'
000E74= 'P='
000E76= $2020  (F8-Start= $E76)
000E78= $2020
000E7A= $2020
000E7C= $2020
000E7E= $0D0A

000E80= 'GE'
000E82= 'AR'
000E84= '= '
000E86= $2020  (Gear-Start= $E86)
000E88= $2020
000E8A= $2020
000E8C= $2020
000E8E= $0D0A
000E90= $0A0A
000E92--------(Data String End)
---------------------------------------------------------------------
```

```
MAIN:
     Link Trap #14 extension table to ROM table
001000 207C00005000     MOVE.L #20480,A0         define starting
                                                  table address
001006 1E3C00FD         MOVE.8 #253,D7           call LINKIT
00100A 4E4E             TRAP   #14
00100C 21C85040         MOVE.L A0,$00005040      move linking
                                                  pointer to end of
                                                  trap extension
                                                  table (points to
                                                  standard ROM table)
001010 1E3C0000         MOVE.B #0,D7             call p-init
001014 4E4E             TRAP   #14
001016 1E3C0001         MOVE.8 #1,D7             call e-init
00101A 4E4E             TRAP   #14


BEGIN LOOP:
     Write Header String to Console
00101C 2A7C00000E00     MOVE.L #3584,A5          starting address
001022 2C7C00000E3E     MOVE.L #3646,A6          ending address
001028 1E3C00F3         MOVE.8 #243,D7           call OUTPUT
00102C 4E4E             TRAP   #14
```

```
     Blank out old values in Data String with spaces
00102E 21FC202020200E46 MOVE.L #538976288,$00000E46
001036 21FC202020200E4A MOVE.L #5389762BB,$00000E4A
00103E 21FC202020200E56 MOVE.L #53B9762BB,$00000E56
001046 21FC202020200E5A MOVE.L #53B9762BB,$00000E5A
00104E 21FC202020200E66 MOVE.L #53B9762BB,$00000E66
001056 21FC202020200E6A MOVE.L #538976288,$00000E6A
00105E 21FC202020200E76 MOVE.L #53B976288,$00000E76
001066 21FC202020200E7A MOVE.L #53B9762BB,$00000E7A
00106E 21FC202020200E86 MOVE.L #53897628B,$00000E86
001076 21FC202020200EBA MOVE.L #53B9762BB,$00000E8A

     Call subroutine GETOATA:
00107E 4EBB1300          JSR.S   $00001300

     Prepare Oata String by converting hex data to decimal
         and then storing results at thelr appropriate
         position in Data String.
                       RACK;
0010B2 4280              CLR.L  00
001084 303B09B0          MOVE.W $00000980,00        get hex value of
                                                     rack from current
                                                     conditions
                                                     structure
001088 2C7C00000E46      MOVE.L #3654,A6            get Rack-Start in
                                                     Data String
00108E 1E3C00EC          MOVE.B #236,07             call HEX2OEC
001092 4E4E              TRAP   #14

                       THSTP;
001094 42B0              CLR.L  DO
001096 303809B2          MOVE.W $000009B2,00        get hex value of
                                                     thstp from current
                                                     conditions
                                                     structure
00109A 2C7C00000E56      MOVE.L #3670,A6            get TH-Start in
                                                     Oata String
0010A0 1E3C00EC          MOVE.B #236,07             call HEX2OEC
0010A4 4E4E              TRAP   #14

                       RGSTP;
0010A6 4280              CLR.L  00
0010A8 30380984          MOVE.W $000009B4,00        get hex value of
                                                     rgstp from current
                                                     conditions
                                                     structure
0010AC 2C7C00000E66      MOVE.L #3686,A6            get RG-Start in
                                                     Data String
0010B2 1E3C00EC          MOVE.8 #236,07             call HEX2OEC
0010B6 4E4E              TRAP   #14
```

```
                      FBSTP:
0010B8 4280                CLR.L   D0
0010BA 303809B6            MOVE.W $000009B6,D0         get hex value of
                                                        fbstp from current
                                                        conditions
                                                        structure
0010BE 2C7C00000E76        MOVE.L #3702,A6             get FB-Start in
                                                        Data String
0010C4 1E3C00EC            MOVE.B #236,D7              call HEX2DEC
0010C8 4E4E                TRAP    #14
                      GEAR:
0010CA 4280                CLR.L   00
0010CC 303809B8            MOVE.W $000009B8,D0         get hex value of
                                                        gear from current
                                                        conditions
                                                        structure
0010D0 2C7C00000E86        MOVE.L #3718,A6             get Gear-Start in
                                                        Data String
0010D6 1E3C00EC            MOVE.B #236,D7              call HEX2DEC
0010DA 4E4E                TRAP    #14
     Write Data String to Console
0010DC 2A7C00000E40        MOVE.L #3648,A5             starting address
0010E2 2C7C00000E92        MOVE.L #3730,A6             ending address
0010E8 1E3C00F3            MOVE.B #243,D7              call OUTPUT
0010EC 4E4E                TRAP    #14

     Call COMPUTE (new desired position) subroutine
0010EE 4EB82000            JSR.S   $00002000
     Set new outputs
0010F2 1E3C0002            MOVE.B #2,D7                call move-throttle
0010F6 4E4E                TRAP    #14
0010F8 1E3C0003            MOVE.B #3,D7                call move-cvt
0010FC 4E4E                TRAP    #14
0010FE 1E3C0004            MOVE.B #4,D7                call set-gear
001102 4E4E                TRAP    #14
001104 1E3C0006            MOVE.B #6,D7                call test-stable
001108 4E4E                TRAP    #14

     Update data structures
          Current == Desired
00110A 4E71                NOP
00110C 4E71                NOP
00110E 4E71                NOP
001110 31F809C209B4        MOVE.W $000009C2,$000009B4 rgstp
001116 31F809C409B8        MOVE.W $000009C4,$000009B8 gear
     Jump to LOOP
00111C 4EF8101C            JMP.S   $0000101C
ENO LOOP:
END MAIN= $1120
```

```
***********************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT O. FUNK
*    OATE:          8/17/85
*    FILE:          getdat.sub.s
*    TYPE:          Sequential Program Execution -- subroutine
*
***********************************************************************
```

SYNOPSIS:

GETDAT is a subroutine called by MAIN. The purpose of the subroutine is to collect all data needed by the program. This data consists of the digital inputs under direct control of the MC68000, and the Analog inputs under the AOAC 1000 control.

Currently only the digital inputs are read, since the desired outputs are determined by interactive user responses.

To Call;

```
        JSR $1300
```

DATA STRUCTURES:

GLOBAL STRUCTURES AFFECTEO:
current external conditions

LOCAL GETOATA SCOPE...
local data allocation ($1200 to $12FF)

```
GETOATA:
     Read external data under direct access of controller
001300 1E3C0005        MOVE.B #5.07              call read-rack
001304 4E4E            TRAP   #14

001306 1E3C0009        MOVE.B #9,07              call read-thstp
00130A 4E4E            TRAP   #14

00130C 1E3C000B        MOVE.B #11,07             call read-fbstp
001310 4E4E            TRAP   #14
001312 4E75            RTS
ENO GETDATA= $1314
```

```
************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT D. FUNK
*    DATE:          8/17/85
*    FILE:          compute.sub.s
*    TYPE:          Sequential Program Execution -- subroutine
*
************************************************************************
```

SYNOPSIS:

COMPUTE is a subroutine called by MAIN. The purpose of this subroutine is to compute the new desired outputs based upon the current operating state of the drive line.

Currently, the subroutine interactively communicates with the user to determine what the new desired operating state should be. In it's final form, this subroutine will implement the optimization algorithm. All inputs to the subroutine will be obtained by the GETDAT subroutine.

To Call;

```
        JSR $2000
```

DATA STRUCTURES:

GLOBAL STRUCTURES AFFECTED:
- console buffer
- desired external conditions

LOCAL COMPUTE SCOPE...
local data allocation ($1500 to $1FFF)

```
-------------------------------------------------------------------
001500= 'EN'            (String #1)
001502= 'TE'
001504= 'R '
001506= 'NE'
001508= 'W '
00150A= 'DA'
00150C= 'TA'
00150E= ': '
001510= $0D0A
001512-------- (String #1 End)
-------------------------------------------------------------------
```

```
-----------------------------------------------------------------
001514= 'TH'            (String #2)
001516= 'ST'
001518= 'P '
00151A= '? '
00151C-------- (String #2 End)
-----------------------------------------------------------------

-----------------------------------------------------------------
00151E= 'RG'            (String #3)
001520= 'ST'
001522= 'P '
001524= '? '
001526-------- (String #3 End)
-----------------------------------------------------------------

-----------------------------------------------------------------
001528= 'GE'            (String #4)
00152A= 'AR'
00152C= ' ?'
00152E= $0020
001530-------- (String #4 End)
-----------------------------------------------------------------
```

```
COMPUTE:
     Write String #1 to Console
002000 2A7C00001500     MOVE.L #5376,A5          starting address
002006 2C7C00001512     MOVE.L #5394,A6          ending address
00200C 1E3C00F3         MOVE.B #243,D7           call OUTPUT
002010 4E4E             TRAP   #14

     Write String #2 to Console
002012 2A7C00001514     MOVE.L #5396,A5          starting address
002018 2C7C0000151C     MOVE.L #5404,A6          ending address
00201E 1E3C00F3         MOVE.B #243,D7           call OUTPUT
002022 4E4E             TRAP   #14

     Get a string from the Console
002024 2A7C00000900     MOVE.L #2304,A5          console buffer
                                                  starting address
00202A 2C4D             MOVE.L A5,A6
00202C 1E3C00F1         MOVE.B #241,D7           call PORTIN1
002030 4E4E             TRAP   #14

     Convert console buffer string to hex and store result
          in Desired Condition Structure
002032 4280             CLR.L  D0
002034 BDCD             CMP.L  A5,A6             skip if null
```

```
002036 670A             BEQ.S  $002042           line
002038 1E3C00E1         MOVE.B #225.07           call GETNUMD
00203C 4E4E             TRAP   #14
00203E 31C009C0         MOVE.W 00,$000009C0      store at Oesired
                                                  THSTP

     Wrlte Strlng #3 to Console
002042 2A7C0000151E     MOVE.L #5406,A5          starting address
002048 2C7C00001526     MOVE.L #5414,A6          ending address
00204E 1E3C00F3         MOVE.B #243,07           call OUTPUT
002052 4E4E             TRAP   #14

     Get a string from the Console
002054 2A7C00000900     MOVE.L #2304,A5          console buffer
                                                  starting address
00205A 2C40             MOVE.L A5,A6
00205C 1E3C00F1         MOVE.B #241,D7           call PORTIN1
002060 4E4E             TRAP   #14

     Convert console buffer string to hex and store results
          in Oesired Condition Structure
002062 4280             CLR.L  00
002064 B0C0             CMP.L  A5,A6             skip if null
002066 670A             BEQ.S  $002072            line
002068 1E3C00E1         MOVE.B #225,07           call GETNUMO
00206C 4E4E             TRAP   #14
00206E 31C009C2         MOVE.W 00,$000009C2      store at Oesired
                                                  RGSTP
     Write String #4 to Console
002072 2A7C00001528     MOVE.L #5416,A5          starting address
002078 2C7C00001530     MOVE.L #5424,A6          ending address
00207E 1E3C00F3         MOVE.B #243,07           call OUTPUT
002082 4E4E             TRAP   #14
     Get a string from the Console
002084 2A7C00000900     MOVE.L #2304.A5          console buffer
                                                  starting address
00208A 2C40             MOVE.L A5,A6
00208C 1E3C00F1         MOVE.B #241,D7           call PORTIN1
002090 4E4E             TRAP   #14
     Convert console buffer string to hex and store result
          in Desired Condition Structure
002092 4280             CLR.L  00
002094 B0CD             CMP.L  A5,A6             skip if null
002096 670A             BEQ.S  $0020A2            line
002098 1E3C00E1         MOVE.B #225,07           call GETNUMO
00209C 4E4E             TRAP   #14
00209E 31C009C4         MOVE.W D0,$000009C4      store at Desired
                                                  GEAR
0020A2 4E75             RTS
END COMPUTE= $20A4
```

```
*********************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT D. FUNK
*    DATE:          8/17/85
*    FILE:          pinit.s
*    TYPE:          Software Interrupt Processing -- level 0
*
*********************************************************************
```

SYNOPSIS:

P-INIT initializes the peripherals which are contained on the 68000 ECB. Configurations are set for the host ACIA the system real time interrupts and the normal shut down interrupt. Auto Vector interrupts are established for these devices and the CPU is set to Supervisory Mode. In addition the short message "P-INIT" is displayed on the Console indicating successful initialization.

TO CALL;

```
          MOVE.B #00,D7
          TRAP   #14
```

DATA STRUCTURES:

GLOBAL STRUCTURES AFFECTED:

none

LOCAL P-INIT SCOPE...

```
-------------------------------------------------------------------
005080= 'P-'            (pinit string)
005082= 'IN'
005084= 'IT'
005086= $0D0A
005088--------- (pinit string End)
-------------------------------------------------------------------
```

P-INIT:

```
     Initialize Auto Vector Interrupts
          Host Interrupt Vector= $7000
006000 21FC000070000078 MOVE.L #28672,$00000078    host interrupt
```

```
     Configure Host ACIA #2
006008 123900010041      MOVE.B $00010041,D1         dumb read on
                                                      receive register
00600E 123900010043      MOVE.B $00010043,D1         dumb read on
                                                      transmit register
006014 13FC009500010041 MOVE.B #149,$00010041        set control
                                                      register format
     Write "P-INIT" string to Console
00601C 2A7C00005080      MOVE.L #20608,A5            starting address
006022 2C7C00005088      MOVE.L #20616,A6            ending address
006028 1E3C00F3          MOVE.B #243,D7              call OUTPUT
00602C 4E4E              TRAP   #14

     Set CPU to Supervisory Mode
00602E 46FC2000          MOVE.W #8192,SR
006032 4E75              RTS
END PINIT= $6034
```

```
************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT D. FUNK
*    DATE:          8/17/85
*    FILE:          einit.s
*    TYPE:          Software Interrupt Processing -- level 1
*
************************************************************************
```

SYNOPSIS:

E-INIT initializes the peripherals which are contained on the expansion bus. Configurations are set for both off board PI/T's to be used for bit I/O and stepper motor controllers. Finally E-INIT calls SETUP which conducts an interactive mode of initializing the physical engine and transmissions.

TO CALL;

```
        MOVE.B #01,D7
        TRAP   #14
```

DATA STRUCTURES:

GLOBAL STRUCTURES AFFECTED:

none

LOCAL E-INIT SCOPE...

```
-----------------------------------------------------------------
0050C8= 'E-'             (einit string #1)
0050CA= 'IN'
0050CC= 'IT'
0050CE= $0D0A
0050D0-------- (einit string #1 End)
-----------------------------------------------------------------

-----------------------------------------------------------------
00508A= $2020            (einit string #2)
00508C= $2020
00508E= 'SE'
005090= 'T '
005092= 'EN'
005094= 'AB'
005096= 'LE'
```

```
005098= 'S '
00509A= 'ON'
00509C= $000A
00509E-------- (einit string #2 End)
--------------------------------------------------------------------

E-INIT:
     Write einit string #1 to Console
006100 2A7C000050C8       MOVE.L #206B0,A5          starting address
006106 2C7C000050D0       MOVE.L #206BB,A6          ending address
00610C 1E3C00F3           MOVE.B #243,07            call OUTPUT
006110 4E4E               TRAP   #14

     Initialize PI/T #1  Base Address= $20000
006112 13FC000000020001 MOVE.B #0,$00020001        Port General
                                                    Control Register
00611A 13FC000000020003 MOVE.B #0,$00020003        Port Service
                                                    Request Register
006122 13FC008000020000 MOVE.B #12B,$0002000D      Port A Control
                                                    Register
00612A 13FC00FF00020005 MOVE.B #255,$00020005      Port A Oata
                                                    Oirection Register
006132 13FC000000020011 MOVE.B #0,$00020011        Port A Data
                                                    Register
00613A 13FC00800002000F MOVE.B #128,$0002000F      Port B Control
                                                    Register
006142 13FC000000020007 MOVE.B #0,$00020007        Port B Oata
                                                    Oirection Register
00614A 13FC000300020009 MOVE.B #3,$00020009        Port C Oata
                                                    Oirection Register
006152 13FC000000020027 MOVE.B #0,$00020027        Counter Preload
                                                    Register High
00615A 13FC000000020029 MOVE.B #0,$00020029        Counter Preload
                                                    Register Mid
006162 13FC00010002002B MOVE.B #1,$0002002B        Counter Preload
                                                    Register Low
00616A 13FC00A700020021 MOVE.B #167,$00020021      Timer Control
                                                    Register

     Initialize PI/T #2  Base Address= $20200
006172 13FC000000020201 MOVE.B #0,$00020201        Port General
                                                    Control Register
00617A 13FC000000020203 MOVE.B #0,$00020203        Port Service
                                                    Request Register
0061B2 13FC000300020209 MOVE.B #3,$00020209        Port C Oata
                                                    Direction Register
00618A 13FC000000020227 MOVE.B #0,$00020227        Counter Preload
                                                    Register High
006192 13FC000000020229 MOVE.B #0,$00020229        Counter Preload
                                                    Register Mid
00619A 13FC00010002022B MOVE.B #1,$0002022B        Counter Preload
```

```
                                                   Register Low
0061A2 13FC00A700020221 MOVE.8 #167,$00020221     Timer Control
                                                   Register
0061AA 13FC000000020205 MOVE.8 #0,$00020205       Port A Data
                                                   Direction Register
006182 13FC00800002020D MOVE.B #128,$0002020D     Port A Control
                                                   Register
0061BA 13FC00FF00020207 MOVE.B #255,$00020207     Port B Data
                                                   Direction Register
0061C2 13FC00B00002020F MOVE.8 #128,$0002020F     Port B Control
                                                   Register
0061CA 13FC001C00020213 MOVE.8 #28,$00020213      Port B Data
                                                   Register

     Write einit string #2 to Console
          prompts user to switch external power "on"
0061D2 2A7C000050BA     MOVE.L #20618,A5          starting address
0061D8 2C7C0000509E     MOVE.L #20638,A6          ending address
0061DE 1E3C00F3         MOVE.8 #243,D7            call OUTPUT
0061E2 4E4E             TRAP   #14

     Complete physical system initialization
0061E4 1E3C000A         MOVE.B #10,D7             call setup
0061EB 4E4E             TRAP   #14
0061EA 4E75             RTS
END E-INIT= $61EC
```

```
*************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECRANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTROR:         KENT D. FUNK
*    DATE:           8/17/85
*    FILE:           thmove.s
*    TYPE:           Software Interrupt Processing -- level 2
*
*************************************************************************
```

SYNOPSIS:

MOVE-TR provides the software interface to the stepper motor driver associated with the engine throttle. The associated hardware requires the following inputs:

1) Direction control (PI/T #1 portC-1)
   CW (active low)
   CCW (active high)
2) Step size (PI/T #1 portC-0)
   For the throttle stepper motor Step Size = RALF (active high)
3) Clock pulse control (PI/T #1 timer)
   One clock pulse per step

MOVE-TR determines the relative number of steps to be moved in a particular direction by knowledge of the Current TRSTP and the Desired THSTP by the following description.

```
     if (desired > actual) {
          move (desired - actual) half steps in CW direction;
          }
     else if ( actual > desired) {
          move (actual - desired) half steps in CCW direction;
          }
     else if ( actual == desired) {
          do nothing;
          }
```

TO CALL:

```
          MOVE.B #02,D7
          TRAP   #14
```

OATA STRUCTURES:

```
     GLOBAL STRUCTURES AFFECTED:
          Program:
               current external conditions
               desired external conditions

     LOCAL MOVE-TH SCOPE...

     ---------------------------------------------------------------------
     005150                  (temporary)
     005151    (4 consecutive bytes for temporary allocation
     005152         for use in byte addressing of final results)
     005153
     ---------------------------------------------------------------------


MOVE-TH:
006200 42B0                CLR.L  DO
006202 42B1                CLR.L  01
006204 303809CO            MOVE.W $000009CO,DO           get desired THSTP
00620B 323B09B2            MOVE.W $000009B2,01           get current THSTP
00620C B041                CMP.W  D1,DO                  check (00 -01) or
                                                          (desired -actual)
00620E 6E1A                BGT.S  $00622A                branch if (desired
                                                          > actual)
006210 674C                BEQ.S  $00625E                branch if (desired
                                                          == actual)
006212 4E71                NOP
006214 4E71                NOP

     Set conditions for actual > desired
006216 92B0                SUB.L  00,01                  compute actual -
                                                          desired (01-00 to
                                                          D1)
00621B 21C15150            MOVE.L 01,$00005150           store results in
                                                          temporary local
                                                          structure
00621C 13FC000300020019 MOVE.B #3,$00020019              set half step, CCW
                                                          direction at port C
006224 6016                BRA.S  $00623C
006226 4E71                NOP
00622B 4E71                NOP

     Set conditions for desired > actual
00622A 9081                SUB.L  01,00                  compute desired -
                                                          actual (DO-01 to
                                                          00)
00622C 21C05150            MOVE.L 00,$00005150           store results in
                                                          temporary local
                                                          structure
```

```
006230 13FC000100020019 MOVE.8 #1,$00020019          set half step, CW
                                                      direction at port C
006238 4E71             NOP
00623A 4E71             NOP
      Move temporary structure to timer
00623C 13F8515100020027 MOVE.8 $00005151,$00020027 store temporary
                                                      2nd byte in timer
                                                      preload register
                                                      (high)
006244 13F8515200020029 MOVE.B $00005152,$00020029 store temporary
                                                      3rd byte in timer
                                                      preload register
                                                      (mid)
00624C 13F851530002002B MOVE.8 $00005153,$00020028 store temporary
                                                      4th byte in timer
                                                      preload register
                                                      (low)
006254 13FC000100020035 MOVE.8 #1,$00020035          start timer
00625C 4E71             NOP
00625E 4E75             RTS
END MOVE-TH= $6260
```

```
**********************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT D. FUNK
*    DATE:           8/17/85
*    FILE:           cvtmove.s
*    TYPE:           Software Interrupt Processing -- level 3
*
**********************************************************************

SYNOPSIS:

     MOVE-CVT provides the software interface to the stepper motor
     driver associated with the CVT transmission.  The associated
     hardware requires the following inputs:

     1)  Direction control (PI/T #2 portC-1)
          CW (active low)
          CCW (active high)
     2)  Step size (PI/T #2 portC-0)
          For the CVT stepper motor Step Size = HALF (active high)
     3)  Clock pulse control (PI/T #2 timer)
          One clock pulse per step

     MOVE-CVT determines the relative number of steps to be moved
     in a particular direction by knowledge of the Current RGSTP
     and the Desired RGSTP by the following description.

          if (desired > actual) {
               move (desired - actual) half steps in CW direction;
               }
          else if ( actual > desired) {
               move (actual - desired) half steps in CCW direction;
               }
          else if ( actual == desired) {
               do nothing;
               }


          TO CALL;
                    MOVE.B #03,D7
                    TRAP   #14
```

DATA STRUCTURES:

GLOBAL STRUCTURES AFFECTED:
Program:
current external conditions
desired external conditions

LOCAL MOVE-CVT SCOPE...

```
----------------------------------------------------------------------
005150                   (temporary)
005151     (4 consecutive bytes for temporary allocation
005152          for use in byte addressing of final results)
005153
----------------------------------------------------------------------
```

```
MOVE-CVT:
006300 42B0              CLR.L   D0
006302 4281              CLR.L   D1
006304 303809C2          MOVE.W  $000009C2,D0        get desired RGSTP
00630B 323B09B4          MOVE.W  $000009B4,D1        get current RGSTP
00630C B041              CMP.W   D1,D0               check (D0-D1) or
                                                      (desired-actual)
00630E 6E12              BGT.S   $006322             branch if (desired
                                                      > actual)
006310 673E              BEQ.S   $006350             branch if (desired
                                                      == actual)

     Set conditions for actual > desired
006312 92B0              SU8.L   D0,01               compute (actual -
                                                      desired) or (D1 -D0
                                                      to D1)
006314 21C15150          MOVE.L  01,$00005150        store results in
                                                      temporary structure
006318 13FC000300020219  MOVE.B  #3,$00020219        set half step, CCW
                                                      direction at port C
006320 600E              8RA.S   $006330

         Set conditions for desired > actual
006322 90B1              SUB.L   D1,D0               compute (desired -
                                                      actual) or (D0 -D1
                                                      to 00)
006324 21C05150          MOVE.L  D0,$00005150        store results in
                                                      temporary structure
006328 13FC000100020219  MOVE.8  #1,$00020219        set half step, CW
                                                      direction at port C
```

```
     Move temporary storage to timer one byte at a time
006330 13F8515100020227 MOVE.8 $00005151,$00020227 store temporary
                                                     2nd byte in timer
                                                     preload register
                                                     (high)
006338 13F8515200020229 MOVE.8 $00005152,$00020229 store temporary
                                                     3rd byte in timer
                                                     preload register
                                                     (mid)
006340 13F8515300020228 MOVE.B $00005153,$0002022B store temporary
                                                     4th byte in timer
                                                     preload register
                                                     (low)
006348 13FC000100020235 MOVE.8 #1,$00020235        start timer
006350 4E75             RTS
END MOVE-CVT= $6352
```

```
*************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT D. FUNK
*    OATE:           8/17/85
*    FILE:           setgear.s
*    TYPE:           Software Interrupt Processing -- level 4
*
*************************************************************************
```

SYNOPSIS:

SET-GEAR provides the software interface to the hardware controller for the Power/Shift transmission. The associated hardware consists of six bit manipulated external control lines via PI/T #1 Port A. The Power/Shift transmission can be shifted from one gear to another by actuation of the appropriate solenoids mounted on the transmission. The following specifications have been established.

Port A assignment;

PA0 == SOL#1
PA1 == SOL#2
PA2 == SOL#3
PA3 == SOL#4
PA4 == SOL#5
PA5 == SOL#6

GEAR SELECTION;

GEAR	SOL#1	SOL#2	SOL#3	SOL#4	SOL#5	SOL#6
NEUTRAL	0	0	0	0	0	0
1ST	1	0	0	0	0	1
2ND	1	0	0	0	1	0
3RD	0	1	0	0	0	1
4TH	0	1	0	0	1	0
5TH	0	0	1	0	0	1
6TH	0	0	1	0	1	0
R1	1	0	0	1	0	0
R2	0	1	0	1	0	0
R3	0	0	1	1	0	0

In addition, a lookup table has been established (see below) which contains the hex representations of Port A for the above listed gears. SET-GEAR determines the appropriate port bit pattern by using the desired gear as the offset in the lookup table, and then writes the pattern to Port A.

TO CALL;

```
        MOVE.B #04,07
        TRAP   #14
```

OATA STRUCTURES:

GLOBAL STRUCTURES AFFECTEO:

Program;

desired external conditions

LOCAL SET-GEAR SCOPE...

```
------------------------------------------------------------------
005144= $00    (N offset= 00)      (pwr/shift lookup table)
005145= $21    (1st offset= 01)
005146= $11    (2nd offset= 02)
005147= $22    (3rd offset= 03)
00514B= $12    (4th offset= 04)
005149= $24    (5th offset= 05)
00514A= $14    (6th offset= 06)
00514B= $09    (R1 offset= 07)
00514C= $0A    (R2 offset= 08)
00514D= $0C    (R3 offset= 09)
00514E-------- (lookup table End)
------------------------------------------------------------------
```

```
SET-GEAR:
006400 4281           CLR.L  D1
006402 323809C4       MOVE.W $000009C4,01        get desired gear
006406 227C00005144   MOVE.L #20804,A1           get base address
                                                  of pwr/shift lookuo
                                                  table
00640C 03C1           AOD.L  01,A1               add deslred gear
                                                  (offset) to base
                                                  address, store
                                                  results in A1
00640E 130100020011   MOVE.B (A1),$00020011      move the byte
                                                  stored at the
                                                  address in A1 to
                                                  PI/T #1 Port A Oata
                                                  Register
006414 4E75           RTS
ENO SET-GEAR= $6416
```

```
*************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARO COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL ANO AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT O. FUNK
*    DATE:           8/17/85
*    FILE:           readrack.s
*    TYPE:           Software Interrupt Processing -- level 5
*
*************************************************************************
```

SYNOPSIS:

REAO-RACK provides the software interface to the 10-bit absolute encoder (Litton 76NB10-5-S-1) which is mechanically attached to the CAT diesel injector pump "rack". Only 8-bits of resolution are achieved, however, due to mechanical linkage. These eight external data lines are accessible via PI/T #1 Port B.

```
TO CALL;
        MOVE.B #05,D7
        TRAP   #14
```

DATA STRUCTURES:

GLOBAL STRUCTURES AFFECTEO:
Program;
current external conditions

LOCAL REAO-RACK SCOPE...
none

```
REAO-RACK:
006500 4280              CLR.L  00
006502 103900020013      MOVE.B $00020013,D0      get data a PI/T #1
                                                   Port B Oata
                                                   Register
006508 31C009B0          MOVE.W 00,$000009B0      store data at RACK
                                                   in current external
                                                   conditions
                                                   structure.
00650C 4E75              RTS
ENO READ-RACK= $650E
```

```
***********************************************************************
*
*
*    ASSEMBLY LISTING FDR MOTOROLA M68000 ECB SINGLE BDARD COMPUTER
*         CVT -- ENGINE PRDJECT -- CONTRDLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHDR:        KENT D. FUNK
*    DATE:          6/17/85
*    FILE:          teststable.s
*    TYPE:          Software Interrupt Processing -- level 6
*
***********************************************************************
```

SYNOPSIS:

TEST-STABLE insures that the drive line actuators are stable before releasing control back to sequential execution.

Due to the slow dynamics of the controller actuators compared to the high speed of the CPU, it is necessary to wait until all actuators have stopped before continuing with the loop. TEST-STABLE checks each of the outputs to determine if they are stopped, thus insuring that the drive line actuators are stahle. This does not insure that the drive line is stable, however, since the dynamics of drive line response will lag the actuators.

To Call;

```
        MOVE.B #6,D7
        TRAP   #14
```

DATA STRUCTURES:

GLOBAL STRUCTURES AFFECTED:
none

LDCAL TEST-STABLE SCOPE...
none

TEST-STABLE:

Wait until throttle stepper motor stops

```
006600 4280            CLR.L   DO
006602 103900020035    MDVE.B  $00020035,DO      get PI/T #1 timer
                                                  status byte
006608 028000000001    AND.L   #I,DO             isolate timer
                                                  empty bit
C0660E 67F2            BEQ.S   $006602           branch up if zero
```

```
     Wait until CVT stepper motor stoos
006610 4280             CLR.L  DO
006612 103900020235     MOVE.B $00020235,DO        get PI/T #2 timer
                                                    status byte
006618 028000000001     AND.L  #1,DO               isolate timer
                                                    empty bit
00661E 67F2             BEQ.S  $006612             branch up if zero
006620 4E75             RTS
END TEST-STABLE= $6622
```

```
**********************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARO COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    OEPARTMENTS OF MECHANICAL ANO AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT O. FUNK
*    DATE:          8/17/85
*    FILE:          regsave.s
*    TYPE:          Software Interrupt Processing -- level 7
*
**********************************************************************
```

SYNOPSIS:

REG-SAVE saves all CPU internal data registers not saved as a part of normal context switching. It is intended to be used as the first procedure to be executed as part of an interrupt handler. Since interrupts occur asynchronously, this mechanisms provides appropriate safeguards against loss of critical data during interrupt exception processing.

TO CALL;

```
MOVE.L 07,$511C
MOVE.B #07,07
TRAP   #14
```

WARNING:

In order to insure that ALL data remains secure, register 07 must be individually saved before calling this routine. This can be accomplished by the above code. In addition, the approoriate Motorola M68000 manuals shuuld be consulted reguarding exception context switching before using this call. It should also be noted that this procedure does not protect against multiple nested interrupts. If this were to occur, critical data will be lost. For the present time, this is acceptable, however, in the future a moving interrupt stack will be used.

DATA STRUCTURES:

GLOBAL STRUCTURES AFFECTED:
none

LOCAL REG-SAVE SCOPE...

```
-----------------------------------------------------------------
005100    (00)          (context save)
```

```
005104     (D1)
005108     (D2)
00510C     (D3)
005110     (D4)
005114     (D5)
005118     (D6)
00511C     (D7)
005120     (A0)
005124     (A1)
005128     (A2)
00512C     (A3)
005130     (A4)
005134     (A5)
005138     (A6)
00513C     (SR)  WORD ONLY
00513C-------- (context save End)
-----------------------------------------------------------------
```

```
REG-SAVE:
006700 21C05100          MOVE.L D0,$00005100          save D0
006704 21C15104          MOVE.L D1,$00005104          save D1
006708 21C25108          MOVE.L D2,$00005108          save D2
00670C 21C3510C          MOVE.L D3,$0000510C          save D3
006710 21C45110          MOVE.L D4,$00005110          save D4
006714 21C55114          MOVE.L D5,$00005114          save D5
006718 21C65118          MOVE.L D6,$00005118          save D6
00671C 21C85120          MOVE.L A0,$00005120          save A0
006720 21C95124          MOVE.L A1,$00005124          save A1
006724 21CA5128          MOVE.L A2,$00005128          save A2
006728 21CB512C          MOVE.L A3,$0000512C          save A3
00672C 21CC5130          MOVE.L A4,$00005130          save A4
006730 21CD5134          MOVE.L A5,$00005134          save A5
006734 21CE5138          MOVE.L A6,$00005138          save A6
006738 40F8513C          MOVE.W SR,$0000513C          save SR
00673C 4E75              RTS
END REG-SAVE= $673E
```

```
*********************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARO COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    OEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT O. FUNK
*    OATE:          8/17/85
*    FILE:          regrestore.s
*    TYPE:          Software Interrupt Processing -- level 8
*
*********************************************************************
```

SYNOPSIS:

REG-RESTORE restores all CPU internal data registers which were saved by REG-SAVE. It is intended to be used as the last procedure to be executed as part of an interrupt handler. This mechanism, in conjunction with REG-SAVE provides appropriate safeguards against loss of critical data during interrupt exception processing.

```
     TO CALL;
               MOVE.B #08,D7
               TRAP   #14
```

WARNING:

In order for this routine to work correctly their must be a correctly installed REG-SAVE preceding it.

DATA STRUCTURES:

```
     GLOBAL STRUCTURES AFFECTEO:
          none

     LOCAL REG-RESTORE SCOPE...
-----------------------------------------------------------------
005100   (00)          (context restore)
005104   (D1)
005108   (02)
00510C   (D3)
005110   (04)
005114   (D5)
005118   (06)
00511C   (D7)
005120   (A0)
```

```
005124    (A1)
005128    (A2)
00512C    (A3)
005130    (A4)
005134    (A5)
005138    (A6)
00513C    (SR)  WORO ONLY
00513C-------- (context restore End)
-------------------------------------------------------------------
```

```
REG-RESTORE:
006800 20385100        MOVE.L $00005100,00        restore D0
006804 22385104        MOVE.L $00005104,01        restore D1
006808 24385108        MOVE.L $00005108,D2        restore D2
00680C 2638510C        MOVE.L $0000510C,D3        restore 03
006810 28385110        MOVE.L $00005110,D4        restore D4
006814 2A385114        MOVE.L $00005114,05        restore D5
006818 2C385118        MOVE.L $00005118,D6        restore 06
00681C 2E38511C        MOVE.L $0000511C,D7        restore D7
006820 20785120        MOVE.L $00005120,A0        restore A0
006824 22785124        MOVE.L $00005124,A1        restore A1
006828 24785128        MOVE.L $00005128,A2        restore A2
00682C 2678512C        MOVE.L $0000512C,A3        restore A3
006830 28785130        MOVE.L $00005130,A4        restore A4
006834 2A785134        MOVE.L $00005134,A5        restore A5
006838 2C785138        MOVE.L $00005138,A6        restore A6
00683C 46F8513C        MOVE.W $0000513C,SR        restore SR
006840 4E75            RTS
ENO REG-RESTORE= $6842
```

```
**************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BDARO COMPUTER
*         CVT -- ENGINE PROJECT -- CDNTRDLLER
*
*    DEPARTMENTS DF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHDR:         KENT D. FUNK
*    DATE:           8/17/85
*    FILE:           readthstp.s
*    TYPE:           Software Interrupt Processing -- level 9
*
**************************************************************************
```

GENERAL:

In order to generate accurate feedback position control for the two stepper motors, incremental encoders have been used. One of the encoders has been mechanically attached to the throttle stepper motor, and the other encoder has been attached to the CVT ring drive. Although this closed loop design would probably not be used in a production unit, it provides the needed repeatability when using expiremental open loop actuators.

The output of the encoders is a pair of square wave pulse trains whose phase indicates relative direction. By correct separation of the two pulse trains, a suitable up/down counter can be driven to give 400 counts/rev resolution. Two 20-bit counters were available (see Spaulding 1985) which correctly read quadrature output. These counters were incorporated in the 68000 off-board bus structure. The counters have suitable controls to multiplex the data to an 8-bit data bus. In addition seven control lines must be supplied. To accommodate the requirements of these devices and others which may be included, an 8-bit external data bus is available at PI/T #2 Port A and an 8-bit external control bus is available at PI/T #2 Port B.

The following external bus definitions apply to the 20-bit up/down counters.

```
     PortA;          (external data bus)
          PA0 == 00
          PA1 == D1
          PA2 == 02
          PA3 == D3
          PA4 == D4
          PA5 == D5
          PA6 == 06
          PA7 == D7
     PortB;          (external control bus)
          PB0 == CLEAR CDUNTER#1    (active high)
```

PB1 == INHIBIT COUNTER#1 (active low)
PB2 == A1
PB3 == A2
PB4 == A3
PB5 == CLEAR COUNTER#2 (active high)
PB6 == INHIBIT COUNTER#2 (active low)

Control of an individual counter is as follows:

Initialization:

1) CLEAR COUNTER and INHIBIT COUNTER lines are brought low.
2) The encoder is driven to "HOME" reference.
3) CLEAR COUNTER is driven high and then low.
4) INHIBIT COUNTER is driven high.

It should be noted that all initialization of both counters is done in the setup routine.

Normal Operation:

Reading the current counter values is accomplished by enabling the desired byte (high, mid, low) and then reading the available data at Port A. Proper enables are accomplished by the following bit map.

Byte Enabled	PB6 INH2	PB5 CL2	PB4 A3	PB3 A2	PB2 A1	PB1 INH1	PB0 CL1	HEX
Counter #1								
low byte	1	0	1	0	0	1	0	$52
mid byte	1	0	1	0	1	1	0	$56
high byte	1	0	1	1	0	1	0	$5A
Counter #2								
low byte	1	0	0	0	0	1	0	$42
mid byte	1	0	0	0	1	1	0	$46
high byte	1	0	0	1	0	1	0	$4A
Disabled	1	0	1	1	1	1	0	$5E

SYNOPSIS:

READ-THSTP provides the software interface to the 100 cycles/rev incremental encoder (BEI L25G-100-ABZ-7400R-S-) which is mechanically attached to the throttle stepper motor. A gear ratio of 2:1 was established between the throttle and encoder and the resolution of the throttle stepper is 400 steps/rev. Therefore, if the counter value is divided by two, the throttle step position is determined. The technique of dividing by two compensates for any backlash in the gear train.

In addition, mechanical considerations indicate that the maximum expected counter value would be less than a 16 bit representation. Therefore, READ-THSTP only reads the low 16 bits of the 20 bit counter.

```
          TO CALL;
                    MOVE.8 #09,D7
                    TRAP   #14


DATA STRUCTURES:

          GLOBAL STRUCTURES AFFECTED:
               Program;
                    current external conditions

          LOCAL READ-THSTP SCOPE...
               none

READ-THSTP:
006900 13FC004200020213 MOVE.8 #66,$00020213      enable counter low
                                                   byte at Port 8 data
                                                   register
006908 4280             CLR.L  D0
00690A 103900020211     MOVE.B $00020211,D0       read counter low
                                                   byte at Port A data
                                                   register
006910 13FC004600020213 MOVE.B #70,$00020213      enable counter mid
                                                   byte at Port 8 data
                                                   register
006918 4281             CLR.L  D1
00691A 123900020211     MOVE.B $00020211,D1       read counter mid
                                                   byte at Port A data
                                                   register
006920 E199             ROL.L  #B,D1              align bytes to
                                                   form correct word
006922 D081             ADD.L  D1,D0
006924 E28B             LSR.L  #1,D0              divide results by
                                                   2
006926 0800000E         BTST   #14,D0             if counter rolled
00692A 6702             8EQ.S  $00692E             under, then set D0
00692C 4280             CLR.L  D0                  to zero.
00692E 31C009B2         MOVE.W D0,$000009B2       store THSTP in
                                                   current conditions
                                                   structure
006932 13FC005E00020213 MOVE.8 #94,$00020213      disable all
                                                   drivers on external
                                                   data bus
00693A 4E75             RTS
END READ-THSTP= $693C
```

```
************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M6BOOO ECB SINGLE BOARO COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    OEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT O. FUNK
*    DATE:          B/17/B5
*    FILE:          readfbstp.s
*    TYPE:          Software Interrupt Processing -- level 11
*
************************************************************************
```

GENERAL:

See REAO-THSTP for general incremental counter considerations.

SYNOPSIS:

READ-FBSTP provides the software interface to the 100 cycles/rev lncremental encoder (BEI L25G-100-ABZ-74OOR-S-) which is mechanically attached to the CVT ring drive. Currently, no exact relation is known between the encoder and the CVT stepper motor. This is due to the tentative development of the ring drive actuator.

Mechanical considerations indicate that the maximum expected counter value would be less than a 16 blt representation. Therefore, READ-FBSTP only reads the low 16 bits of the 20 bit counter.

```
TO CALL;
          MOVE.B #11,O7
          TRAP   #14
```

OATA STRUCTURES:

GLOBAL STRUCTURES AFFECTEO:
- Program;
  - current external conditions

LOCAL REAO-FBSTP SCOPE...
- none

```
REAO-FBSTP:
006950 13FC005200020213 MOVE.B #B2,$00020213      enable counter low
                                                   byte at Port B data
                                                   register
```

```
006958 4280             CLR.L  D0
00695A 103900020211     MOVE.B $00020211,D0        read counter low
                                                    byte at Port A data
                                                    register
006960 13FC005600020213 MOVE.B #86,$00020213       enable counter mid
                                                    byte at Port B data
                                                    register
006918 4281             CLR.L  D1
00696A 123900020211     MOVE.B $00020211,D1        read counter mid
                                                    byte at Port A data
                                                    register
006970 E199             ROL.L  #8,D1               align bytes to
                                                    form correct word
006972 D081             ADD.L  D1,D0
006974 4E71             NOP

006976 0800000E         BTST   #14,D0              if counter rolled
00697A 6702             BEQ.S  $00697E              under, then set D0
00697C 4280             CLR.L  D0                   to zero.

00697E 31C009B6         MOVE.W D0,$000009B6        store FBSTP in
                                                    current conditions
                                                    structure
006982 13FC005E00020213 MOVE.B #94,$00020213       dlsable all
                                                    drivers on external
                                                    data bus
00698A 4E75             RTS
END READ-FBSTP= $698C
```

```
*************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M6B000 ECB SINGLE BOARO COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    OEPARTMENTS OF MECHANICAL ANO AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT D. FUNK
*    OATE:          6/17/B5
*    FILE:          setup.s
*    TYPE:          Software Interrupt Processing -- level 10
*
*************************************************************************
```

SYNOPSIS:

SETUP conducts an interactive mode of initializing the physical engine and transmissions. This procedure is executed only once and must be preformed after all computer hardware has been initialized (p-init and e-init).

To Call;

```
        MOVE.B #10,07
        TRAP   #14
```

OATA STRUCTURES:

GLOBAL STRUCTURES AFFECTEO:
- Program;
  - console buffer
  - current external conditions
  - desired external conditions

LOCAL SETUP SCOPE...

```
-----------------------------------------------------------------
50A0= $20202020             (setup string #1)
50A4= $20202020
50AB= 'CO'
50AA= 'NT'
50AC= 'IN'
50AE= 'UE'
50B0= '? '
50B2-------- (setup string #1 end)
-----------------------------------------------------------------

-----------------------------------------------------------------
50B4= $20202020             (setup string #2)
50BB= 'ST'
```

```
50BA= 'AR'
50BC= 'T '
50BE= 'EN'
50C0= 'GI'
50C2= 'NE'
50C4= $0D0A
50C6-------- (setup string #2 end)
--------------------------------------------------------------------
```

```
SETUP:
     Write setup string #1 to Console
006A00 2A7C000050A0       MOVE.L #20640,A5          starting address
006A06 2C7C000050B2       MOVE.L #2065B,A6          ending address
006A0C 1E3C00F3           MOVE.B #243,D7            call OUTPUT
006A10 4E4E               TRAP   #14

     Wait for user Response
006A12 2A7C00000900       MOVE.L #2304,A5           console buffer
                                                     starting address
006A1B 2C4D               MOVE.L A5,A6
006A1A 1E3C00F1           MOVE.B #241,07            call PORTIN1
006A1E 4E4E               TRAP   #14

     Move throttle 500 steps up
006A20 13FC000100020019 MOVE.B #1,$00020019         select half steps.
                                                     CW rotate at PI/T#1
006A2B 13FC000000020027 MOVE.B #0,$00020027         timer high
006A30 13FC000100020029 MOVE.B #1,$00020029         timer mid
006A3B 13FC00F40002002B MOVE.B #244,$0002002B       timer low
006A40 13FC000100020035 MOVE.B #1,$00020035         start timer
     Wait till throttle stops
006A4B 103900020035       MOVE.B $00020035,00       timer status byte
006A4E 02B000000001       AND.L  #1,00               isolate time out
                                                     bit
006A54 67F2               BEQ.S  $006A4B            branch up if zero

     Search down the throttle 2 steps at a time.
          After every two steps check for home switch detect.
          Insure switch reading by 4 consecutive positive reads.
006A56 13FC000300020019 MOVE.B #3,$00020019         select half step,
                                                     CCW rotate at PI/T#1
006A5E 13FC000000020027 MOVE.B #0,$00020027         timer high
006A66 13FC000000020029 MOVE.B #0,$00020029         timer mid
006A6E 13FC00020002002B MOVE.B #2,$0002002B         timer low
006A76 13FC000100020035 MOVE.B #1,$00020035         start timer
006A7E 123C0000           MOVE.B #0,01              D1 is event
                                                     detection counter
006AB2 103900020035       MOVE.B $00020035,00       get timer status
006ABB 02B000000001       AND.L  #1,D0              isolate time out
                                                     bit
```

```
006A8E 67F2               BEQ.S   $006AB2            branch up if zero
006A90 363CFFFF           MOVE.W #-1,D3              PAUSE for
006A94 57CBFFFE           DBEQ.L D3,$006A94           a moment
006A9B 103900020019       MOVE.B $00020019,D0        get home detect
                                                      input
006A9E 028000000010       AND.L   #16,D0             isolate throttle
                                                      home bit
006AA4 67B0               BEQ.S   $006A56            branch up if zero
006AA6 363CFFFF           MDVE.W #-1,D3              pause for a while
006AAA 57CBFFFE           DBEQ.L D3,$006AAA           to let home
006AAE 363CFFFF           MOVE.W #-1,D3               switch debounce.
006AB2 57CBFFFE           DBEQ.L D3,$006AB2
006AB6 06010001           ADD.B   #1,D1              increment event
                                                      detection counter
006ABA 0C010004           CMP.B   #4,D1              check if we have
                                                      four consecutive
                                                      reads
006ABE 66D8               BNE.S   $006A98            branch uo if not

      Throttle is now at home position
          Now reset 20-bit throttle position counter to zero
006AC0 13FC001D00020213 MOVE.B #29,$00020213         set CLEAR COUNTER
                                                      high
006ACB 13FC001C00020213 MOVE.B #28,$00020213         set CLEAR CDUNTER
                                                      low
006AD0 13FC001E00020213 MDVE.B #30,$00020213         enable counter

      Move throttle up 900 steps
006AD8 13FC000100020019 MDVE.B #1,$00020019          select half step,
                                                      CW rotation @ PI/T#1
006AE0 13FC000000020027 MDVE.B #0,$00020027          timer high
006AE8 13FC000400020029 MDVE.B #4,$00020029          timer mid
006AF0 13FC004C0002002B MDVE.B #76,$0002002B         timer low
006AF8 13FC000100020035 MOVE.B #1,$00020035          start timer
006B00 4E71               NDP
006B02 4E71               NDP
006B04 4E71               NDP
006B06 31FC044C09C0       MOVE.W #1100,$000009C0     update desired
                                                      position structure
                                                      THSTP

      Write setup string #2 to Console
006B0C 2A7C000050B4       MOVE.L #20660,A5           starting address
006B12 2C7C000050C6       MDVE.L #2067B,A6           ending address
006B18 1E3C00F3           MOVE.B #243,D7             call OUTPUT
006B1C 4E4E               TRAP    #14

      Write setup string #1 to Console
006B1E 2A7C000050A0       MOVE.L #20640,A5           starting address
006B24 2C7C000050B2       MOVE.L #2065B,A6           ending address
```

```
006B2A 1E3C00F3          MOVE.B #243,D7          call DUTPUT
006B2E 4E4E              TRAP   #14

     Wait for user Response
006B30 2A7C00000900      MOVE.L #2304,A5         console buffer
                                                  starting address
006B36 2C4D              MDVE.L A5,A6
006B3B 1E3C00F1          MOVE.B #241,D7          call PORTIN1
006B3C 4E4E              TRAP   #14

     Move Ring Position 500 steps up
006B3E 13FC000100020219 MDVE.B #1,$00020219      select half steps,
                                                  CW rotate at PI/T#2
006B46 13FC000000020227 MDVE.B #0,$00020227      timer high
006B4E 13FC000100020229 MOVE.B #1,$00020229      timer mid
006B56 13FC00F40002022B MDVE.B #244,$0002022B    timer low
006B5E 13FC000100020235 MOVE.B #1,$00020235      start timer

     Wait till Ring stops
006B66 103900020235      MOVE.B $00020235,D0     timer status byte
006B6C 028000000001      AND.L  #1,D0            isolate time out
                                                  bit
006B72 67F2              BEQ.S  $006B66          branch up if zero

     Search down the ring travel 2 steps at a time.
          After every two steps check for home switch detect.
          Insure switch reading by 4 consecutive positive reads.
006B74 13FC000300020219 MOVE.B #3,$00020219      select half step,
                                                  CCW rotate at PI/T#2
006B7C 13FC000000020227 MDVE.B #0,$00020227      timer high
006BB4 13FC000000020229 MOVE.B #0,$00020229      timer mid
006B8C 13FC00020002022B MDVE.B #2,$0002022B      timer low
006B94 13FC000100020235 MDVE.B #1,$00020235      start timer
006B9C 123C0000          MDVE.B #0,D1            D1 is event
                                                  detection counter
006BA0 103900020235      MDVE.B $00020235,D0     get timer status
                                                  byte
006BA6 02B000000001      AND.L  #1,D0            isolate time out
                                                  bit
006BAC 67F2              BEQ.S  $006BA0          branch up if zero
006BAE 363CFFFF          MDVE.W #-1,03           PAUSE for
006BB2 57CBFFFE          DBEQ.L D3,$006BB2        a moment
006BB6 103900020219      MOVE.B $00020219,D0     get home detect
                                                  input
006BBC 02B000000010      ANO.L  #16,D0           isolate ring home
                                                  bit
006BC2 67B0              BEQ.S  $006B74          branch up if zero
006BC4 363CFFFF          MOVE.W #-1,D3           pause for awhile to
006BC8 57CBFFFE          DBEQ.L $006BCB           let home switch
006BCC 363CFFFF          MOVE.W #-1,03            debounce.
006BD0 57CBFFFE          DBEQ.L $006BD0
```

```
006B04 06010001         A00.B  #1,01             increment event
                                                  detection counter
006B08 0C010004         CMP.B  #4,01             check if we have
                                                  four consecutive
                                                  reads
006B0C 66D8             BNE.S  $006BB6           branch up if not

     CVT rings are now at home position
          Reset and initialize fb-stp counter.
006BDE 13FC003E00020213 MOVE.B #62,$00020213
006BE6 13FC001E00020213 MOVE.B #30,$00020213
006BEE 13FC005E00020213 MOVE.B #94,$00020213

     Update the rest of the global data structures
006BF6 31FC000009B4     MOVE.W #0,$000009B4      current RGSTP
006BFC 31FC000009C2     MOVE.W #0,$000009C2      desired RGSTP
006C02 31FC000009BB     MOVE.W #0,$000009B8      current gear
006C08 31FC000009C4     MOVE.W #0,$000009C4      desired gear
006C0E 31FC000009B6     MOVE.W #0,$000009B6      current FBSTP
006C14 4E75             RTS
END SETUP= $6C16
```

```
*************************************************************************
*
*
*    ASSEMBLY LISTING FOR MOTOROLA M68000 ECB SINGLE BOARD COMPUTER
*         CVT -- ENGINE PROJECT -- CONTROLLER
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT D. FUNK
*    DATE:          6/17/85
*    FILE:          hostint.s
*    TYPE:          Hardware Interrupt Processing -- ADAC 1000
*
*************************************************************************
```

SYNOPSIS:

HOST-INT provides the interrupt handler for all MC68000 to ADAC communications. In it's final form, this routine will control communications in both directions. Currently, however, the routine only provides for data to be passed from the MC68000 to the ADAC. This has proved sufficient for drive line mapping. Once the implementation of the control algorithm is in place, bi-directional data flow will be needed.

Control of data flow is accomplished by message passing. If the ADAC desires updated information from the MC68000, it writes a 'R' for request to ACIA #2. When the character is caught, a priority 6 hardware interrupt is generated. The existence of this interrupt prompts the MC68000 to enter exception processing using HOST-INT as the handler. Once it has been determined that a valid request has been made, HOST-INT updates the current conditions structure and proceeds to form an ASCII string from the data. Notice that a tag 'S' is placed at the beginning of the string. This allows the ADAC to know that the MC68000 is "sending" data to a valid request. These tags, ie 'S' and 'R', allow for adequate error checking and prevents handling spurious interrupts.

DATA STRUCTURES:

GLOBAL STRUCTURES AFFECTEO:
    Program;
        current external conditions

LOCAL HOST-INT SCOPE...

```
------------------------------------------------------------------
5800= 'S '                  (host response string)
5802= $2020 (rack-start)
```

```
5804= $2020
5806= $2020
5808= $2020
580A= $2020 (thstp-start)
580C= $2020
580E= $2020
5810= $2020
5812= $2020 (rgstp-start)
5814= $2020
5816= $2020
5818= $2020
581A= $2020 (fbstp-start)
581C= $2020
581E= $2020
5820= $2020
5822= $2020 (gear-start)
5824= $2020
5826= $2020
5828= $2020
582A= $000A
582C-------- (host response string end)
------------------------------------------------------------
```

```
HOST-INT:
007000 13FC001500010041 MOVE.B #21,$00010041       shut off ACIA
                                                    interrupt bit
007008 21C7511C         MOVE.L D7,$0000511C        save register D7
00700C 1E3C0007         MOVE.B #7,D7               call SAVE-REG
007010 4E4E             TRAP   #14
007012 1A3900010041     MOVE.B $00010041,D5        dumb read on
                                                    status register
007018 4285             CLR.L  D5
00701A 1A3900010043     MOVE.B $00010043,D5        get a character
007020 0205007F         AND.B  #127,D5             mask high bit
007024 0C050052         CMP.B  #82,D5              is it an `R'
007028 6704             BEQ.S  $00702E             branch to REQUEST:
00702A 4EF870EC         JMP.S  $000070EC           jump to CLEAR:

REQUEST:
     Blank out host response string
00702E 207C00005800     MOVE.L #22528,A0           base address
007034 30FC5320         MOVE.W #21280,(A0)+        'S '
007038 20FC20202020     MOVE.L #538976288,(A0)+    spaces
00703E 20FC20202020     MOVE.L #538976288,(A0)+    spaces
007044 20FC20202020     MOVE.L #538976288,(A0)+    spaces
00704A 20FC20202020     MOVE.L #538976288,(A0)+    spaces
007050 20FC20202020     MOVE.L #538976288,(A0)+    spaces
007056 20FC20202020     MOVE.L #538976288,(A0)+    spaces
00705C 20FC20202020     MOVE.L #538976288,(A0)+    spaces
007062 20FC20202020     MOVE.L #538976288,(A0)+    spaces
```

```
007068 20FC20202020      MDVE.L #538976288,(A0)+     spaces
00706E 20FC20202020      MDVE.L #538976288,(A0)+     spaces
007074 30FC200A          MOVE.W #8202,(A0)+          space-LF

     Update current conditions structure
007078 1E3C0005          MOVE.8 #5,D7                call READ-RACK
00707C 4E4E              TRAP   #14
00707E 1E3C0009          MOVE.8 #9,D7                call READ-THSTP
007082 4E4E              TRAP   #14
007084 1E3C0008          MDVE.8 #11,D7               call READ-F8STP
007088 4E4E              TRAP   #14

     Get RACK reading, convert, and store in host response string.
00708A 303809B0          MOVE.W $00000980,D0         rack reading
00708E 2C7C00005802      MDVE.L #22530,A6            position in host
                                                      string
007094 1E3C00EC          MDVE.8 #236,D7              call HEX2DEC
007098 4E4E              TRAP   #14

     Get TH-STP reading, convert, and store in host response string.
00709A 30380982          MDVE.W $000009B2,D0         th-stp reading
00709E 2C7C0000580A      MOVE.L #22538,A6            position in host
                                                      string
0070A4 1E3C00EC          MOVE.8 #236,D7              call HEX2DEC
0070A8 4E4E              TRAP   #14

     Get RG-STP reading, convert, and store in host response string.
0070AA 30380984          MDVE.W $00000984,D0         rg-stp reading
0070AE 2C7C00005812      MDVE.L #22546,A6            position in host
                                                      string
007084 1E3C00EC          MDVE.8 #236,07              call HEX2DEC
0070B8 4E4E              TRAP   #14

     Get F8-STP reading, convert, and store in host response string.
00708A 30380986          MOVE.W $00000986,D0         fb-stp reading
00708E 2C7C0000581A      MDVE.L #22554,A6            position in host
                                                      string
0070C4 1E3C00EC          MDVE.8 #236,D7              call HEX2DEC
0070C8 4E4E              TRAP   #14

     Get GEAR reading, convert, and store in host response string.
0070CA 303809B8          MOVE.W $00000988,00         gear reading
0070CE 2C7C00005822      MOVE.L #22562,A6            position in host
                                                      string
0070D4 1E3C00EC          MDVE.8 #236,D7              call HEX2DEC
007008 4E4E              TRAP   #14


     Write host response string out to ACIA.
0070DA 2A7C00005800      MOVE.L #22528,A5            string starting
                                                      address
```

```
0070E0 2C7C0000582C      MOVE.L #22572,A6           string ending
                                                     address + 1
0070E6 1E3C00F2          MOVE.B #242,07             call OUTPUT21
0070EA 4E4E              TRAP   #14

CLEAR:
0070EC 103900010041      MOVE.B $00010041,00        dumb read on ACIA
                                                     status register
                                                     clears interrupt
                                                     conditions
0070F2 08000000          BTST   #0,D0               test if clear
0070F6 6708              BEQ.S  $007100             branch on clear to
                                                     OONE:
0070F8 1A3900010043      MOVE.B $00010043,D5        dumb read on
                                                     receive register.
0070FE 60EC              BRA.S  $0070EC             branch to CLEAR:

DONE:
007100 1E3C0008          MOVE.B #8,D7               call REG-RESTORE
007104 4E4E              TRAP   #14
007106 13FC009500010041 MOVE.B #149,$00010041      enable ACIA
                                                     interrupts
00710E 4E73              RTE                        return from
                                                     exception
END HOST-INT= $7110
```

APPENDIX E

ADAC 1000 HOST DEVELOPMENT SOFTWARE LISTINGS

```
************************************************************************
*
*
*    C-SOURCE LISTING FOR ADAC 1000 COMPUTER
*         CVT -- ENGINE PROJECT -- MC68000 DEVELOPMENT
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:        KENT D. FUNK
*    DATE:          4/5/85
*    FILE:          transfer.c
*
************************************************************************

/* This file contains a program to upload or download programs
between a UNIX machine and the MC68000 ECB.  Both S-records and
listings can be handled this way. */

/* Too complile and link:   --> cc transfer.c
     Too run:  --> transfer       */


#include <stdio.h>

FILE *fp, *fopen();
char line[80];


/******                    MAIN()                               ******/
main()
{
     char file[20];
     int  direct;

     printf("Enter filename --> ");
     fflush(stdout);
     fgets(line, 80, stdin);
     sscanf(line, "%s", file);

     printf("\nEnter option.\n");
     printf("Dump=1, Load=2\n");
     fgets(line, 80, stdin);
     sscanf(line, "%d", &direct);

     if (direct == 1) {
exit:
          printf("Waiting for your exit.\n");
          fgets(line, 80, stdin);
               /* exiting character sent by tutor= $01 */
          if (*line != 1)
               goto exit;
```

```
          dump(file);
          fclose(fp);
          return;
     }

     if (direct == 2) {
          printf("Waiting for your exit.\n");
start:
          fgets(line, 80, stdin);
               /* Tutor uses "VE2 ;=READY" or
                          "L02 ;=READY" to verify
                    and load S-Record Format */
          if (*line != 'R')
               goto start;
          load(file);
          fclose(fp);
          return;
     }
     printf("Incorrect option specified.\n\n");
}


/******                        DUMP()                        ******/
/*
Dump Routine.
     The dump routine reads the standard input and writes
     out to the named file. */

dump(fname)
char *fname;
{
     fp= fopen(fname, "w");
loop:
     fgets(line, 80, stdin);
     if (*line == 0)
          goto loop;
          /* To stop input, return to shell and type "x" */
     if (*line == 'x')
          return;
     fprintf(fp, "%s", line);
     goto loop;
}


/******                        LOAD()                        ******/
/*
Load Routine.
     The load routine reads from the named file and writes
     to the standard output.  */
```

```
load(fname)
char *fname;
{
     fp= fopen(fname, "r");
loop:
     fgets(line, 80, fp);
     if (feof(fp))
          return;
     printf("%s", line);
     goto loop;
}


/* NOTICE:  For these routines to work properly, the tutor escape
character must be ^A (default). and the trailing character must
be $0A,<lf>.  The line feed must be specified in the
initialization proceedures as follows:
     TUTOR 1.3 > MM 4EA
     0004EA   18 ?0A.<cr>
*/
```

```
***********************************************************************
*
*
*    C-SOURCE LISTING FOR ADAC 1000 COMPUTER
*         CVT -- ENGINE PROJECT -- MC68000 OEVELOPMENT
*
*    DEPARTMENTS OF MECHANICAL AND AGRICULTURAL ENGINEERING
*
*    AUTHOR:         KENT D. FUNK
*    DATE:           4/5/85
*    FILE:           comment.c
*
***********************************************************************

/* This file contains a program to comment MC68000 programs which
have been dumoed onto a UNIX system using "transfer" */

/* Too comolile and link:   --> cc -o comment comment.c */

/* In order to use this comment editor, the assembly listings
MUST first be packed.  This can be done by creating two files:

FILE #1:  oack
     x $1 <pedit

FILE #2:  oedit
     g/    /s//  /
     g/   /s// /
     w
     q

Then the source files may be packed by:
     --> pack <filename>

Then the source files may be commented by:
     --> comment <filename>
*/


#include <stdio.h>
#include <ctype.h>


FILE *r, *w, *fopen();

/******                        MAIN()                           ******/
main(argc, argv)
int argc;
char **argv;
{
     char buff[200], blank[80];
```

```
     char line1[200], line2[200], line3[200], file[20];
     char *ptr1, *ptr2, *ptr3, *ptr4;
     int len1, len2, spaces, index;

     if (argc != 2) {
          fprintf(stderr, "Usage:  comment <filename>.\n");
          exit(1);
     }
     strcpy(file, argv[1]);
     spaces= 57;
     ptr3= &blank[0];
     while (spaces) {
          *ptr3= ' ';
          ptr3++;
          spaces--;
     }


restart:
     r= fopen(file, "r");      /* open file for read
                                 at top of file */

     w= fopen("comment.edt", "w"); /* opens writing file */
loop:
     ptr1= line1;
     ptr3= line3;
     index= 200;
     while (index) {
          *ptr1= 0;
          *ptr3= 0;
          ptr1++;
          ptr3++;
          index--;
     }

     strcpy(line3, blank);
     fgets(line1, 150, r);     /* get a line from the opened file */
     if (feof(r))
          printf("End of File\n");
tryagain:
     ptr2= line2;
     index= 200;
     while (index) {
          *ptr2= 0;
          ptr2++;
          index--;
     }

     printf("%s", line1);
     printf("?> ");
     fflush(stdout);
```

```
     fgets(line2, 150, stdin);


     switch (*line2) {
          case 10:
               fprintf(w, "%s", line1);
               break;

          case '?':
               printf("Available editor commands are:\n");
               printf("\tl (line of text)\tinserts the line of\n");
               printf("\t\t\t\ttext above the current line.\n");
               printf("\ta (line of text)\tappends the line of\n");
               printf("\t\t\t\ttext to the current line.\n");
               printf("\td\t\t\tdeletes the current line.\n");
printf("\tw\t\t\tsaves the file. resume at the top of the file.\n");
               printf("\tq\t\t\tquits comment.\n");
               printf("\t<ret>\t\t\tskips to next line.\n");
               printf("\t?\t\t\tprints this list.\n");
               goto tryagain;
               break;

          case 'd':
               break;

          case 'i':
               line2[0]= ' ';
               fprintf(w, "%s", line2);
               goto tryagain;
               break;

          case 'a':
               line2[0]= ' ';
               len1= strlen(line1);
               spaces= 57 - len1;
               if (spaces < 0) {
                    printf("Do not comment this line.\n");
                    goto tryagain;
                    }
               ptr1 = &line1[len1-1];
               while(spaces){
                    *ptr1= ' ';
                    ptr1++;
                    spaces--;
                    }
               ptr1--;
               len2= strlen(line2);
               if (len2 <= 20) {
                    ptr2= &line2[0];
                    while (*ptr2) {
                         *ptr1= *ptr2;
```

```
                    ptr1++;
                    ptr2++;
                    }
               fprintf(w, "%s", line1);
               goto loop;
          }
          ptr3= &line3[57];
          ptr2= &line2[20];
          while (isspace(*ptr2) == 0) {
               ptr2--;
          }
          ptr4= &line2[0];
          while (ptr4 < ptr2) {
               *ptr1= *ptr4;
               ptr4++;
               ptr1++;
          }
          *ptr1= '\n';
          fprintf(w, "%s", line1);
          while (*ptr2) {
               *ptr3= *ptr2;
               ptr3++;
               ptr2++;
          }
addon:
          ptr1= line1;
          index = 200;
          while (index) {
               *ptr1= 0;
               ptr1++;
               index--;
          }
          strcpy(line1, line3);
          len1= strlen(line1);
          if (len1 <= 77) {
               fprintf(w, "%s", line1);
               goto loop;
          }
          ptr3= line3;
          index= 200;
          while (index) {
               *ptr3= 0;
               ptr3++;
               index--;
          }
          strcpy(line3, blank);
          ptr1= &line1[77];
          ptr3= &line3[57];
          while (isspace(*ptr1) == 0) {
               ptr1--;
          }
```

```
                ptr4= ptr1;
                while (*ptr1) {
                     *ptr3= *ptr1;
                     ptr3++;
                     ptr1++;
                }
                *ptr4= '\n';
                ptr4++;
                *ptr4= 0;
                fprintf(w, "%s", line1);
                goto addon;

           case 'w':
                while (feof(r) == 0) {
                     fprintf(w, "%s", line1);
                     fgets(line1, 150, r);
                }
                fclose(w);
                fclose(r);
                w= fopen(file, "w");
                r= fopen("comment.edt", "r");
writback:
                fgets(line1, 150, r);
                if (feof(r) == 0) {
                     fprintf(w, "%s", line1);
                     goto writback;
                }
                fclose(w);
                fclose(r);
                goto restart;
                break;

           case 'q':
                fclose(w);
                fclose(r);
                unlink("comment.edt");
                printf("Comment Exiting\n");
                exit();
                break;

           default:
                printf("Editor options (i a d w q <ret> ?) \n");
                goto tryagain;
      }
      goto loop;
}
```

APPENDIX F

DATA SHEETS

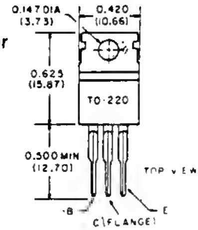
SK3180

263

NPN Si AF
Darlington Transistor

P_T	65 W
I_C	10 A
V_{CBO}	80 V
V_{CEO}	80 V
V_{EBO}	5 V
h_{FE}	20,000
f_T	20 MHz

1

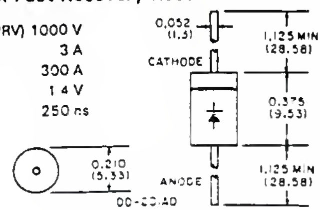


Made in Malaysia

SK5040

Silicon Fast Recovery Roctifier

V_{RRM}(PRV)	1000 V
$I_o(I_{FAV})$	3 A
I_{FSM}	300 A
V_F	1.4 V
t_{rr}	250 ns



Made in U.S.A.

MOTOROLA Semiconductors

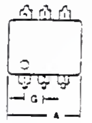
4N30
4N31
4N32, 4N32A
4N33

INFRARED LIGHT EMITTING DIODE PHOTO DARLINGTON TRANSISTOR COUPLED PAIR

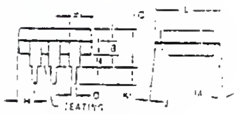
NPN PHOTOTRANSISTOR AND PN INFRARED EMITTING DIODE

. . . Gallium Arsenide LED optically coupled to a Silicon Photo Darlington Transistor designed for applications requiring electrical isolation, high-current transfer ratios, small package size and low cost; such as interfacing and coupling systems, phase and feedback controls, solid-state relays and general-purpose switching circuits.

- High Isolation Voltage V_{ISO} = 7500 V (Min)
- High Collector Output Current @ I_F = 10 mA — I_C = 50 mA (Min) — 4N32,33; 10 mA (Min) — 4N29,30; 5.0 mA (Min) — 4N31
- Economical, Compact, Dual-In-Line Package
- Excellent Frequency Response — 30 kHz (Typ)
- Fast Switching Times @ I_C = 50 mA: t_{on} = 0.6 µs (Typ); t_{off} = 17 µs (Typ) — 4N29,30,31; 45 µs (Typ) — 4N32,33
- 4N29A, 4N32A are UL Recognized — File Number E54915



STYLE 1
PIN 1. ANODE
2. CATHODE
3. NC
4. EMITTER
5. COLLECTOR
6. BASE



NOTES:
1. LEADS WITHIN 0.13 mm (0.005) DIAMETER OF TRUE POSITION AT SEATING PLANE AT MAXIMUM MATERIAL CONDITION.
2. DIMENSION L TO CENTER OF LEADS WHEN FORMED PARALLEL.

DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	[illegible]	[illegible]	[illegible]	[illegible]
B	[illegible]	[illegible]	[illegible]	[illegible]
C	[illegible]	[illegible]	[illegible]	[illegible]
D	[illegible]	[illegible]	[illegible]	[illegible]
F	[illegible]	[illegible]	[illegible]	[illegible]
G	[illegible]	[illegible]	[illegible]	[illegible]
H	[illegible]	[illegible]	[illegible]	[illegible]
J	[illegible]	[illegible]	[illegible]	[illegible]
K	[illegible]	[illegible]	[illegible]	[illegible]
L	[illegible]	[illegible]	[illegible]	[illegible]
M	[illegible]	[illegible]	[illegible]	[illegible]
N	[illegible]	[illegible]	[illegible]	[illegible]

CASE 730-01

MAXIMUM RATINGS (T_A = 25°C unless otherwise noted)

Rating	Symbol	Value	Unit
INFRARED-EMITTING DIODE MAXIMUM RATINGS			
Reverse Voltage	V_R	3.0	Volts
Forward Current — Continuous	I_F	80	mA
Forward Current — Peak (Pulse Width = 300 µs, 2.0% Duty Cycle)	I_F	3.0	Amp
Total Power Dissipation @ T_A = 25°C, Negligible Power in Transistor	P_D	150	mW
Derate above 25°C		2.0	mW/°C
PHOTOTRANSISTOR MAXIMUM RATINGS			
Collector-Emitter Voltage	V_{CEO}	30	Volts
Emitter-Collector Voltage	V_{ECO}	5.0	Volts
Collector-Base Voltage	V_{CBO}	30	Volts
Total Power Dissipation @ T_A = 25°C	P_D	150	mW
Negligible Power in Diode, Derate above 25°C		2.0	mW/°C
TOTAL DEVICE RATINGS			
Total Device Dissipation @ T_A = 25°C, Equal Power Dissipation in Each Element	P_D	250	mW
Derate above 25°C		3.3	mW/°C
Operating Junction Temperature Range	T_J	-55 to +100	°C
Storage Temperature Range	T_{stg}	-55 to +150	°C
Soldering Temperature (10 s)	—	260	°C

FIGURE 1 — MAXIMUM POWER DISSIPATION

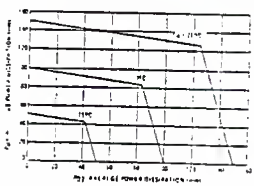


Figure 1 is based upon using limit values in the equation:

$T_{J1} = T_A + R_{\theta JA}(P_{D1} + K_\theta P_{D2})$

where

T_{J1} Junction Temperature (100°C)
T_A Ambient Temperature
$R_{\theta JA}$ Junction to Ambient Thermal Resistance (500°C/W)
P_{D1} Power Dissipation in One Chip
P_{D2} Power Dissipation in Other Chip
K_θ Thermal Coupling Coefficient (20%)

Example:
With P_{D1} = 90 mW in the LED @ T_A = 50°C, the Darlington P_D (P_{D2}) must be less than 50 mW.

Characteristic	Symbol	Min	Typ	Max	Unit
*Reverse Leakage Current (V_R = 3.0 V, R_L = 1.0 M ohms)	I_R	–	0.05	100	µA
*Forward Voltage (I_F = 50 mA)	V_F	–	1.2	1.5	Volts
Capacitance (V_R = 0 V, f = 1.0 MHz)	C	–	150	–	pF

PHOTOTRANSISTOR CHARACTERISTICS (T_A = 25°C and I_F = 0 unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
*Collector-Emitter Dark Current (V_{CE} = 10 V, Base Open)	I_{CEO}	–	–	100	nA
*Collector-Base Breakdown Voltage (I_C = 100 µA, I_E = 0)	BV_{CBO}	30	–	–	Volts
*Collector-Emitter Breakdown Voltage (I_C = 100 µA, I_B = 0)	BV_{CEO}	30	–	–	Volts
*Emitter-Collector Breakdown Voltage (I_E = 100 µA, I_B = 0)	BV_{ECO}	5.0	–	–	Volts
DC Current Gain (V_{CE} = 5.0 V, I_C = 500 µA)	h_{FE}	–	5000	–	–

COUPLED CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
*Collector Output Current (1) (V_{CE} = 10 V, I_F = mA, I_B = 0) 4N32, 4N33	I_C	50	–	–	mA
4N29, 4N30		10	–	–	
4N31		5.0	–	–	
Isolation Surge Voltage (2, 5) (60 Hz ac Peak, 5 Seconds)	V_{ISO}	7500	–	–	Volts
*4N29, 4N32		2500	–	–	
*4N30, 4N31, 4N33		1500	–	–	
Isolation Resistance (2) (V = 500 V)	–	–	10^{11}	–	Ohms
*Collector-Emitter Saturation Voltage (1) (I_C = 2.0 mA, I_F = 8.0 mA) 4N31	$V_{CE(sat)}$	–	–	1.2	Volts
4N29, 4N30, 4N32, 4N33		–	–	1.0	
Isolation Capacitance (2) (V = 0, f = 1.0 MHz)	–	–	0.8	–	pF
Bandwidth (3) (I_C = 2.0 mA, R_L = 100 ohms, Figures 6 and 8)	–	–	30	–	kHz

SWITCHING CHARACTERISTICS (Figures 7 and 9), (4)

Characteristic	Symbol	Min	Typ	Max	Unit
Turn-On Time (I_C = 50 mA, I_F = 200 mA, V_{CC} = 10 V)	t_{on}	–	0.8	5.0	µs
Turn-Off Time (I_C = 50 mA, I_F = 200 mA, V_{CC} = 10 V) 4N29, 30, 31	t_{off}	–	17	40	µs
4N32, 33		–	45	100	

*Indicates JEDEC Registered Data.
(1) Pulse Test: Pulse Width = 300 µs, Duty Cycle ≤ 2.0%.
(2) For this test, LED pins 1 and 2 are common and phototransistor pins 4, 5, and 6 are common.
(3) I_F adjusted to yield I_C = 2.0 mA and i_c = 2.0 mA P-P at 10 kHz.
(4) t_d and t_r are inversely proportional to the amplitude of I_F; t_s and t_f are not significantly affected by I_F.
(5) Isolation Surge Voltage, V_{ISO}, is an internal device dielectric breakdown rating.

DC CURRENT TRANSFER CHARACTERISTICS

FIGURE 2 — 4N29, 4N30, 4N31

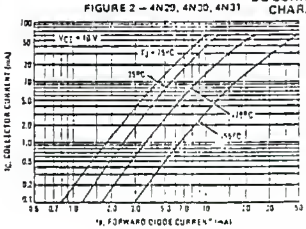
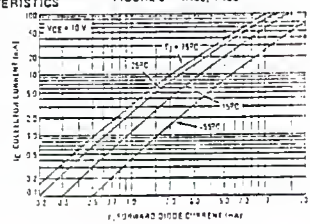


FIGURE 3 — 4N32, 4N33



TYPICAL ELECTRICAL CHARACTERISTICS
(Printed Circuit Board Mounting)

FIGURE 4 – DIODE FORWARD CHARACTERISTIC

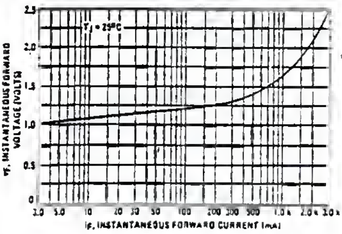


FIGURE 5 – COLLECTOR-EMITTER CUTOFF CURRENT

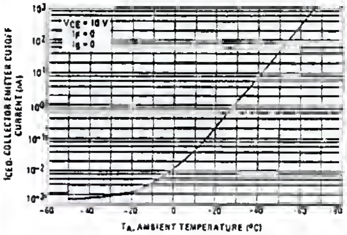


FIGURE 6 – FREQUENCY RESPONSE

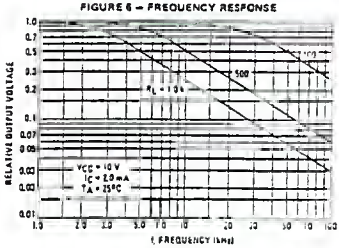
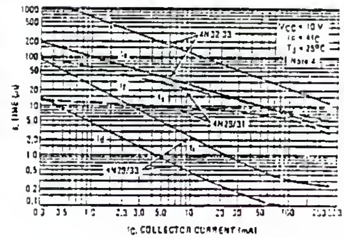


FIGURE 7 – SWITCHING TIMES



4N25

FIGURE 8 – FREQUENCY RESPONSE TEST CIRCUIT

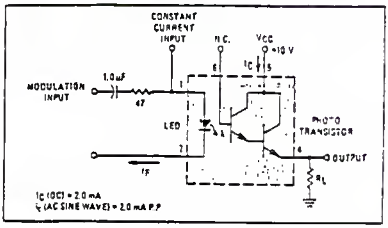
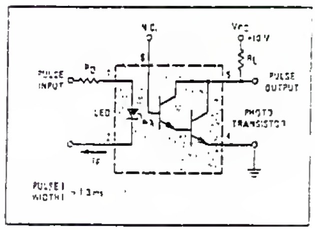


FIGURE 9 – SWITCHING TIME TEST CIRCUIT



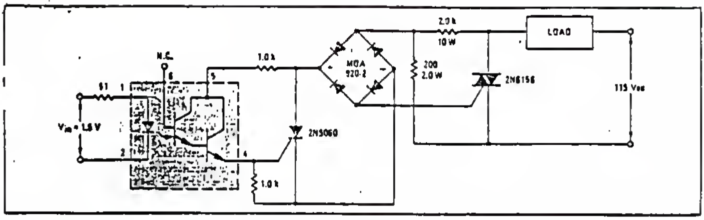


FIGURE 11 – AC SOLID STATE RELAY

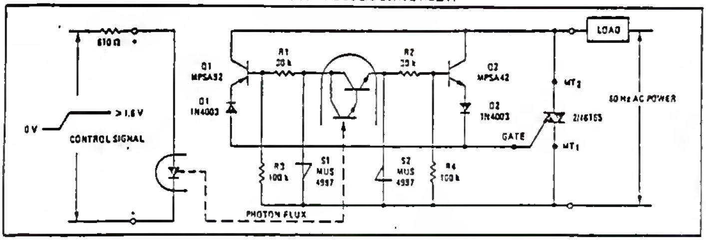


FIGURE 12 – OPTICALLY COUPLED ONE SHOT

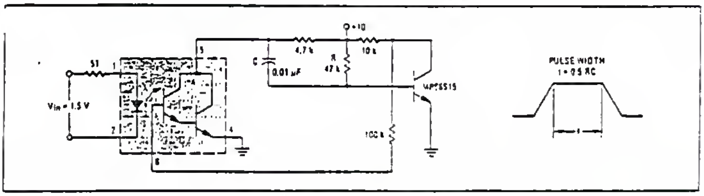
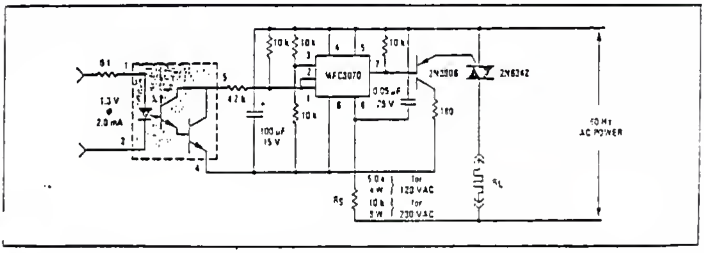


FIGURE 13 – ZERO VOLTAGE SWITCH



MOTOROLA
SEMICONDUCTORS
P.O. BOX 20912 • PHOENIX, ARIZONA 85036

Specifications and Applications Information

SAA1042
SAA1042A

Possible Substitutes

UDN-2949E

Sprague UCN-4202A

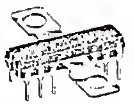
STEPPER MOTOR DRIVER

SILICON MONOLITHIC
INTEGRATED CIRCUIT

STEPPER MOTOR DRIVER

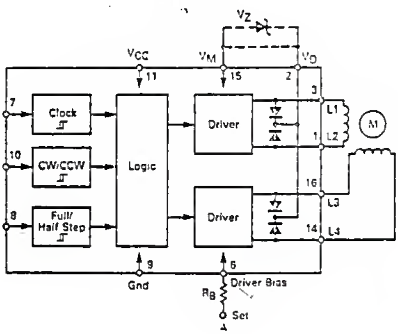
The SAA1042 drives a two-phase stepper motor in the bipolar mode. The device contains: three input stages, a logic section and two output stages.

- Drive Stages Designed for Motors: 6.0 V and 12 V: SAA1042
  24 V: SAA1042A
- 500 mA/Coil Drive Capability
- Built-In Clamp Diodes for Overvoltage Suppression
- Wide Logic Supply Voltage Range
- Accepts Commands for CW/CCW and Half/Full Step Operation
- Inputs Compatible with Popular Logic Families: MOS, TTL, DTL
- Set Input Defined Output State
- Drive Stage Bias Adaptable to Motor Power Dissipation for Optimum Efficiency

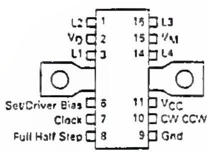


PLASTIC PACKAGE
CASE 721-02

FIGURE 1 — SAA1042 BLOCK DIAGRAM



PIN ASSIGNMENT



(Top View)

Note: Case heat sink is electrically connected to ground (Pin 9) through the die substrate.

…IMUM RATINGS (T_A = 25°C unless otherwise stated)

Rating	Symbol	SAA1042	SAA1042A	Unit
Clamping Voltage (Pins 1, 3, 14 & 16)	V_{clamp}	20	30	V
…ver Voltage (V_{OV} = V_{clamp} − V_M)	V_{OV}	6.0		V
…pply Voltage	V_{CC}	20	30	V
Switching or Motor Current/Coil	I_M	500		mA
Input Voltage (Pins 7, 8 & 10)	V_{in} clock V_{in} Full/Half V_{in} CW/CCW	V_{CC}		V
Power Dissipation Derate above T_A = 25°C Thermal Resistance, Junction to Air Thermal Resistance, Junction to Case	P_D* $1/\theta_{JA}$ θ_{JA} θ_{JC}	2.0 20 50 8.0		W mW/°C °C/W °C/W
Operating Junction Temperature Range	T_J	−30 to +125		°C
Storage Temperature Range	T_{stg}	−65 to +150		°C

*The power dissipation, P_D, of the circuit is given by the supply voltage, V_M and V_{CC}, and the motor current, I_M, and can be determined from Figures 3 and 5. $P_D = P_{drive} + P_{logic}$.

ELECTRICAL CHARACTERISTICS (T_A = +25°C)

Characteristic	Pin	Symbol	V_{CC}	Min	Typ	Max	Unit
Supply Current	11	I_{CC}	5.0 V 20 V	— —	— —	3.5 6.5	mA
Motor Supply Current ($I_{Pin\ 6}$ = −400 µA, Pins 1, 3, 14, 16 Open) V_M = 6.0 V V_M = 12 V V_M = 24 V	15	I_M	 5.0 V 5.0 V 5.0 V	 — — —	 25 30 40	 — — —	mA
Input Voltage — High State	7, 8, 10	V_{IH}	5.0 V 10 V 15 V 20 V	2.0 7.0 10 14	— — — —	— — — —	V
Input Voltage — Low State	7, 8, 10	V_{IL}	5.0 V 10 V 15 V 20 V	— — — —	— — — —	0.0 1.5 2.5 3.5	V
Input Reverse Current — High State (V_{in} = V_{CC})	7, 8, 10	I_{IR}	5.0 V 10 V 15 V 20 V	— — — —	— — — —	2.0 2.0 3.0 5.0	µA
Input Forward Current — Low State (V_{in} = Gnd)	7, 8, 10	I_{IF}	5.0 V 10 V 15 V 20 V	−10 −23 −40 −55	— — — —	— — — —	µA
Output Voltage — High State (V_M = 12 V) I_{out} = −500 mA I_{out} = −50 mA	1, 3, 14, 16	V_{OH}	5.0 to 20 V	 — —	 V_M − 2.0 V_M − 1.2	 — —	V
Output Voltage — Low State I_{out} = 500 mA I_{out} = 50 mA	1, 3, 14, 16	V_{OL}	5.0 to 20 V	 — —	 0.7 0.2	 — —	V
Output Leakage Current (V_M = V_D = V_{clamp} max.) Pin 6: Open	1, 3, 14, 16	I_{OR}	5.0 to 20 V	−100	—	—	µA
Clamp Diode Forward Voltage (Drop at I_M = 500 mA)	2	V_F	—	—	2.5	3.5	V
Clock Frequency	7	f_c	5.0 to 20 V	0	—	50	kHz
…ock Pulse Width	7	t_w	5.0 to 20 V	10	—	—	µs
…t Pulse Width	6	t_s	—	10	—	—	µs
Set Control Voltage — High State Low State	6	—	—	V_M —	— —	— 0.5	V

INPUT/OUTPUT FUNCTIONS

Clock — (Pin 7) This input is active on the positive edge of the clock pulse and accepts Logic '1' input levels dependant on the supply voltage and includes hysteresis for noise immunity.

CW/CCW — (Pin 10) This input determines the motor's rotational direction. When the input is held low, (OV, see the electrical characteristics) the motor's direction is nominally clockwise (CW). When the input is in the high state, Logic '1,' the motor direction will be nominally counter clockwise (CCW), depending on the motor connections.

Full/Half Step — (Pin 8) This input determines the angular rotation of the motor for each clock pulse. In the low state the motor will make a full step for each applied clock pulse, while in the high state, the motor will make half a step.

V_D — (Pin 2) This pin is used to protect the outputs (1, 3, 14, 16) where large positive spikes occur due to switching the motor coils. The maximum allowable voltage on these pins is the clamp voltage (V_{clamp}). Motor performance is improved if a zener diode is connected between Pin 2 and Pin 15 as shown in Figure 1.

The following conditions have to be considered when selecting the zener diode:

$V_{clamp} = V_M + 6.0\ V$

$V_Z = V_{clamp} - V_M - V_F$*

where: V_F = clamp diodes forward voltage drop (see Figure 4)

* V_{clamp}:
  $\leq$ 20 V for SAA1042
  $\leq$ 30 V for SAA1042A

Pins 2 and 15 can be linked, in this case $V_Z = 0$ V.

Set/Bias Input — (Pin 6) This input has two functions:

The resistor R_B adapts the drivers to the motor current.

A pulse via the resistor R_B sets the outputs (1, 3, 14, 16) to a defined state.

The resistor R_B can be determined from the graph of Figure 2 according to the motor current and voltage. Smaller values of R_B will increase the power dissipation of the circuit and larger values of R_B may increase the saturation voltage of the driver transistors.

When the "set" function is not used, terminal A of the resistor R_B must be grounded. When the set function is used, terminal A has to be connected to an open-collector (buffer) circuit. Figure 7 shows this configuration. The buffer circuit (off-state) has to sustain the motor voltage V_M. When a pulse is applied via the buffer and the bias resistor R_B:

During the pulse duration, the motor driver transistors are turned off.

After elapsing the pulse, the outputs will have defined states.

Figure 6 shows the timing diagram.

Figure 7 illustrates a typical application in which the SAA1042 drives a 12 V stepper motor with a current consumption of 200 mA/coil.

A bias resistor (R_B) of 56 kΩ is chosen according to Figure 2.

The maximum voltage permitted at the output pin is V_M + 6.0 V (see the Maximum Ratings). In this application V_M = 12 V, therefore the maximum voltage is 18 V. The outputs are protected by the internal diodes and an external zener connected between Pins 2 and 15.

From Figure 4, it can be seen that the voltage drop across the internal diodes is about 1.7 V at 200 mA. This results in a zener voltage between Pins 2 and 15 of:

$$V_Z = 6.0\ V - 1.7\ V = 4.3\ V.$$

To allow for production tolerances and a safety margin, a 3.9 V zener has been chosen for this example.

The clock is derived from the line frequency which is phase locked by the MC14046B and the MC14024.

The voltage on the clock input, is normally low (Logic '0'). The motor steps on the positive going transition of the clock pulse.

A Logic '0' applied to the Full/Half input, Pin 8, operates the motor in the Full Step mode. A Logic '1' on this input will result in the Half Step mode. The logic level state on the CW-CCW input, Pin 10, and the connection of the motor coils to the outputs determines the rotational direction of the motor.

These two inputs should be biased to a Logic '0' or '1' and not left floating. In the event of non-use, they should be tied to ground or the logic supply line, V_{CC}.

The output drivers can be set to a fixed operating point by use of the Set input and a bias resistor R_B. A positive pulse to this input turns the drivers off and sets the logic state of the outputs.

After the negative going transition of the Set pulse and until the first positive going transition of the clock, the outputs will be:

$$L1 = L3 = \text{high and } L2 = L4 = \text{low}.$$

(See Figure 6, the timing diagram).

The Set input can be driven by a MC14007B or a transistor whose collector resistor is R_B. If the input is not used, the 'bottom' of R_B must be grounded.

The total power dissipation of the circuit can be determined from Figures 3 and 5.

$$P_D = 0.9\ W + 0.03\ W = 0.93\ W.$$

This results in a junction to ambient temperature, without a heatsink of:

$$T_J - T_A = 50°C/W \times 0.93\ W = 49°C.$$

or a maximum ambient temperature of 76°C. For operation at elevated temperatures a heatsink is required.

FIGURE 2 — BIAS RESISTOR R_B versus MOTOR CURRENT

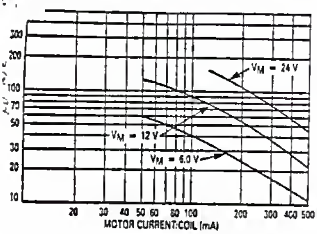


FIGURE 3 — DRIVE STAGE POWER DISSIPATION

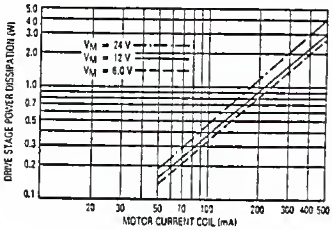


FIGURE 4 — CLAMP DIODE FORWARD CURRENT versus FORWARD VOLTAGE

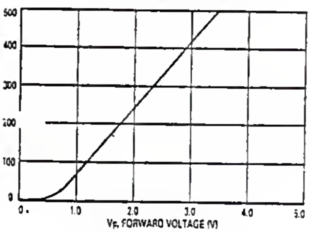


FIGURE 5 — POWER DISSIPATION versus LOGIC SUPPLY VOLTAGE

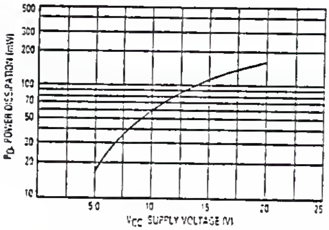


FIGURE 6 — TIMING DIAGRAM

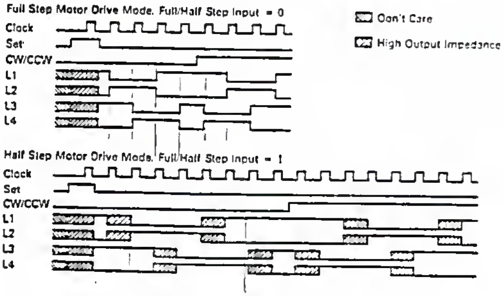
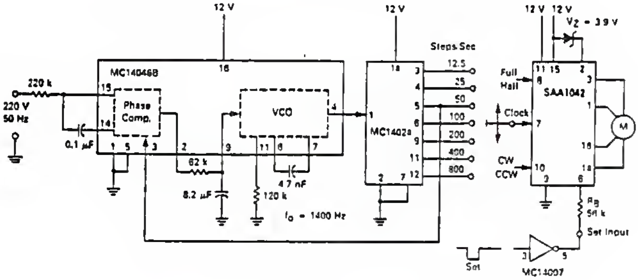
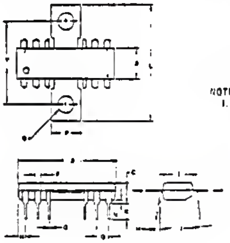


FIGURE 7 — TYPICAL APPLICATION
SELECTABLE STEP RATES WITH THE TIME BASE DERIVED FROM THE LINE FREQUENCY



PACKAGE DIMENSIONS

PLASTIC PACKAGE
CASE 721-02



NOTES
1. DIMENSION "L" TO CENTER OF LEADS WHEN FORMED PARALLEL.

DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	20.70	21.34	0.815	0.840
B	6.10	6.60	0.240	[illegible]
C	4.06	4.57	0.160	0.180
D	0.43	0.56	0.017	0.022
F	1.02	1.52	0.040	0.060
G	2.41	2.67	[illegible]	[illegible]
H	1.32	1.83	0.052	0.072
J	0.33	0.46	0.013	[illegible]
K	3.33	3.94	0.133	0.155
L	25.15	27.54	[illegible]	[illegible]
M	—	10°	—	10°
N	0.51	1.02	[illegible]	0.040
P	6.27	6.53	0.247	0.257
Q	3.44	3.73	0.135	0.147
S	[illegible]	[illegible]	[illegible]	[illegible]
T	[illegible]	[illegible]	0.540	[illegible]

REVISIONS

LTR	DESCRIPTION	CHECK	DATE	APPV.
A	PILOT RELEASE PER EO # WP 0291	EG	9-27-79	MAB
B	INCORPORATED EO WP 0432	EG	2-4-80	MCB

C+H Sales
att DAN DALEY
917 E COLORADO
Pasadena Ca 91107

20-2235 D700 E.1.1

P&I

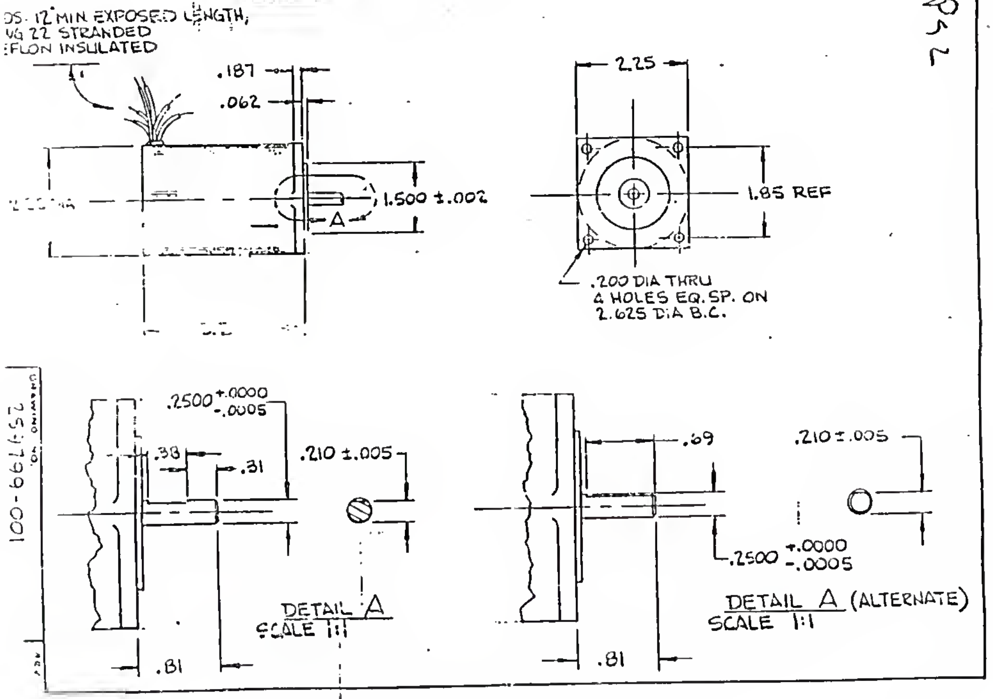
FLNS

E.1.7

PILOT RELEASE

SUPPLEMENTARY INFORMATION				Dataproducts
FIRST USED ON MODEL	DR	A. BURKE	4-79	
J30	CHK	EO	9-27-79	TITLE MOTOR SPECIFICATION, STEPPING MOTOR
	ORIG		9/3/79	
NEXT ASSEMBLY	APP			
704194-100	APP			
	MATL			SIZE: A · CODE IDENT NO: 19790 · DRAWING NO: 259194-001

MOTOR CONFIGURATION



VENDOR: SIGMA INSTRUMENTS, INC. - BRAINTREE, MASS. 02184
MODEL NO: 20-2235D200-E1.6 WITH MODIFIED SHAFT

GENERAL SPECIFICATIONS

TYPE: Permanent magnet rotor
NO. OF PHASES: Two (4 or 8 step switching sequence)
STEP ANGLE: 1.8°
ANGULAR ACCURACY: ±3% of one step, no load, after any number of steps ±.054°
AMBIENT OPERATING TEMPERATURE: -20°C to 50°C without heat sink 122°F
MAXIMUM CASE TEMPERATURE: 100°C
INSULATION: NEMA Class B
INSULATION REISTANCE: 1,000 M @ 500VDC & 25°C
DIELECTRIC STRENGTH:
Between windings and frame: 1,000 Vrms @ 60 Hz
Between windings: 400 Vrms @ 60 Hz
THERMAL RESISTANCE (degrees centigrade per watt)
a. FREE AIR MODE: TBD
b. INFINITE HEATSINK MODE: TBD

1 step = 1.8°

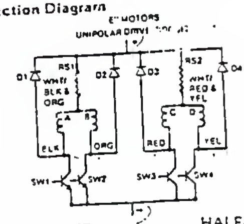
PERFORMANCE SPECIFICATIONS		
ITEM	PARAMETER	VALUE
1		
2	HOLDING TORQUE	120 oz. in.
3	DETENT TORQUE	5.0/0.04 oz-in/Nm
4	PHASE CURRENT (UNIPOLAR)	2.3 amps
-	—	—
-	—	—
5	PHASE RESISTANCE	1.6 ohms
6	PHASE INDUCTANCE	4.2 mH
7	ROTOR INTERTIA	1. [illegible] 0.026 oz-in^2/10^{-3} kgm^2
8	WEIGHT	2.3 lbs (1.04 kg)
9	NUMBER OF LEADS	6 (SEE CONNECTION DIAGRAM FOR COLORS)

4V., 2A.

DRAWING NO.	REV.
259799-001	A
SHEET 3 OF 4 SHEETS	

Pg 4

Connection Diagram



1.8°/STEP

FULL-STEP

STEP	COIL A	COIL B	COIL C	COIL D
1	1	0	0	1
2	1	0	1	0
3	0	1	1	0
4	0	1	0	1
1	1	0	0	1

9 5 6 10 9

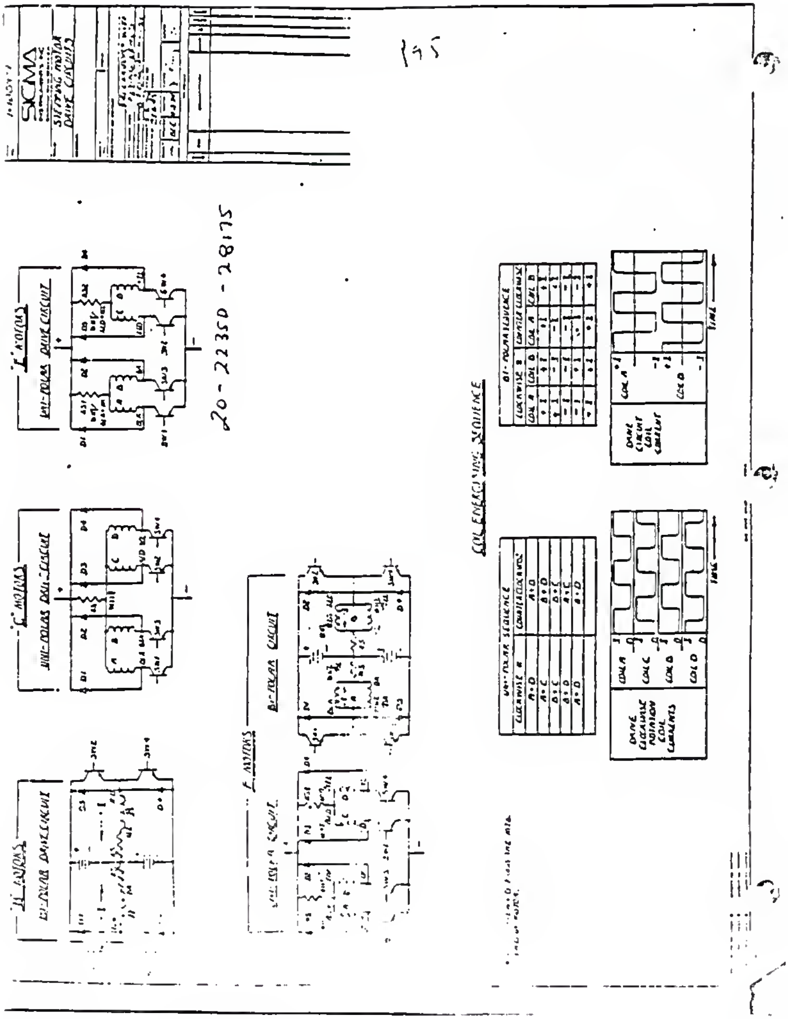
HALF-STEP

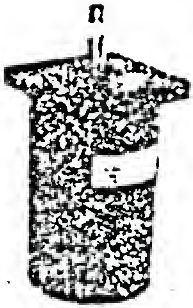
STEP	COIL A	COIL B	COIL C	COIL D
[illegible]	1	0	0	1
2	1	0	0	0
3	1	0	1	0
4	0	0	1	0
5	0	1	1	0
6	0	1	0	0
7	0	1	0	1
8	0	0	0	1
1	1	0	0	1

9 1 5 4 6 2 10 8 9

25V @ 1.5A/winding

DRAWING NO. 259779-001

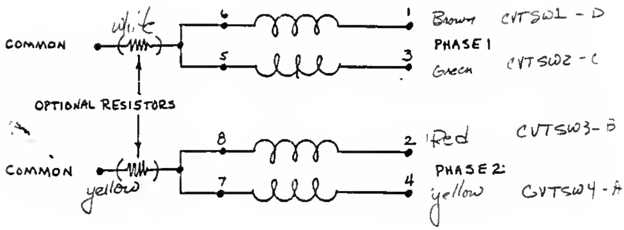

SIGMA
COIL ENERGIZING SEQUENCE



LARGE SIGMA STEPPING MOTOR, #21-4270D-28408, equivalent to Sigma P/N 21-4270D-200 FO3. Permanent magnet rotor. 8 leads. Torque @ 50 PPS - 1000 oz/in. Holding torque 1150 oz/in. Detent torque 22 oz/in. Current per phase 7.6 amps. Nominal phase resistance .3 ohm. Two phase. 1.8° steps. 200 steps per revolution. Any voltage from 3 to 80 VDC can be used with the appropriate external limiting resistor. Can be used with unipolar or bipolar type drive. Dimensions: 4.2" diameter x 7.0" long. Shaft: .624" diameter x 2.3" long. Front mounting flange 4.375" square with 4 mounting holes. Comes complete with specification sheets and wiring diagrams.

CONNECTION DIAGRAM FOR SIGMA STEPPER MOTOR
P/N 21-4270D-200F03

TERMINAL 1	BLACK
TERMINAL 6	BLACK/WHITE
TERMINAL 2	RED
TERMINAL 8	RED/WHITE
TERMINAL 3	ORANGE
TERMINAL 5	ORANGE/WHITE
TERMINAL 4	YELLOW
TERMINAL 7	YELLOW/WHITE



MODEL 76

LOW COST, OPTICAL ABSOLUTE, SHAFT POSITION ENCODER

FEATURES

- Choice of 3 Code Formats
- Solid State Illumination Source
- Simple Design • High Reliability
- Choice of 10 Resolutions
- DTL and TTL Compatible Outputs
- 2 Mounting Configurations
- 2 Shaft Sizes

APPLICATIONS

- NC Machine Tools
- Computing Scales
- Process Control
- Divider Heads
- Plotters • Printers
- Antennas
- Navigation Systems

MODEL 76 has been engineered to provide the potential user the maximum flexibility in selecting the physical and electrical characteristics dictated by the application. There is the choice of two mounting configurations with the input/output connector mounted on the end or side of the housing. Two input shaft styles, three code formats to choose from—Gray Code, Natural Binary and 8421 Binary Coded Decimal—and 10 standard resolutions. Resolutions up to 10 bits in Gray or Natural Binary and several resolutions in BCD available (example 180 and 360).

MODEL 76 only requires a single 5 VDC power supply for operation. The outputs are fully buffered to provide direct DTL and TTL compatibility.

PHYSICAL CHARACTERISTIC OPTIONS

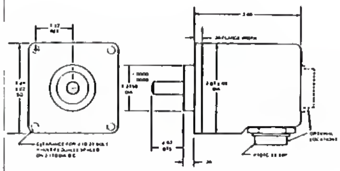


FIGURE 1

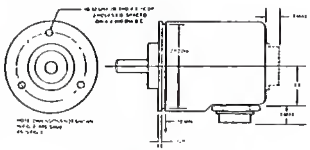


FIGURE 2

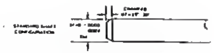


FIGURE 3

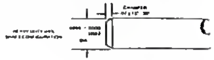


FIGURE 4

USE THIS BLOCK DIAGRAM

TO ORDER

Standard options available are illustrated by the block nomenclature diagram below. When ordering, indicate options desired by completing Model Number with Designation from options tables

76	SHAFT AND BEARING SIZE	CODE FORMAT	RESOLUTION	MOUNTING CONFIGURATION	CONNECTOR LOCATION CONFIGURATION	CONFIGURATION 1-STANDARD
	USE DESIGNATION "HD" FOR HEAVY DUTY VERSION LEAVE BLANK FOR STD CONFIG	USE DESIGNATION FROM TABLE I	USE DESIGNATION FROM TABLE II OR TABLE III	USE DESIGNATION FROM TABLE IV	USE DESIGNATION FROM TABLE V	OTHER DESIGNATIONS FOR SPECIAL CONFIGURATIONS ARE ASSIGNED BY THE FACTORY

TABLES OF OPTIONS

TABLE I

CODE FORMAT OPTIONS	
FORMAT	DESIGNATION
Gray Code	GC
NATURAL BINARY	NB
8421 BCD	BD

TABLE II

RESOLUTION OPTIONS	
These resolutions are available with Gray Code and Natural Binary Models	
RESOLUTION	DESIGNATION
64	06
128	07
256	08
512	09
1024	10

TABLE III

RESOLUTION OPTIONS	
These resolutions are available with 8421 BCD Models	
RESOLUTION	DESIGNATION
100	100
180	180
200	200
360	360
400	400
1000	1000

TABLE IV

MOUNTING CONFIGURATION OPTIONS	
CONFIGURATION	DESIGNATION
Per Fig. 1	1
Per Fig. 2	2
Per Fig. 1 with Shaft Seal	4
Per Fig. 2 with Shaft Seal	5

TABLE V

CONNECTOR LOCATION OPTIONS	
CONFIGURATION	DESIGNATION
At End of Housing	E
On Side of Housing	S

ADDITIONAL OPTIONS

The following non-standard options are available on Special Order:

- Line Driver Outputs
- Serialized Outputs
- Extra Low Torque Bearings
- Other Shaft Configurations
- Special Code Formats
- Other Resolutions

MODEL 76

CONTINUED

GENERAL SPECIFICATIONS

MECHANICAL

PHYSICAL CHARACTERISTICS
Per Fig 1 with optional mounting configurations per Fig. 2 or 3

WEIGHT.
20 oz maximum

MOMENT OF INERTIA
.0004 Oz. In. Sec.^{2}

SHAFT LOADING
Standard Shaft. 30 lbs maximum
Heavy Duty Shaft. 70 lbs maximum

SHAFT ROTATION
Continuous and reversible, CW
Rotation viewing shaft end produces ascending count

ANGULAR ACCELERATION
10^6 radians/sec^2 maximum

SLEWING SPEED
5000 RPM maximum (See note 1)

BEARING LIFE
$\frac{16 \times 10^6}{RPM}$ Hrs. minimum

STARTING TORQUE (See Note 3)
Standard Shaft — No Seal 1 0 oz in max
Standard Shaft — With Seal 5 0 oz in max
Heavy Duty Shaft. 8 0 oz in max

ELECTRICAL

CODE FORMAT
Optional — Gray Code, Natural Binary or B421 Binary Coded Decimal

RESOLUTION
Optional — See Tables II and III

INPUT
5 0±2% VDC at 300 mA maximum

OUTPUT LOGIC LEVELS
Logic '1' Vcc with 4 7 K max Ohm "source" impedance
Logic '0' 0 5 VDC max with 3 mA "sink" current (GC)
0 5 VDC max with 10 mA "sink" current (NB & BD)

ILLUMINATION SOURCE
Type Solid state (GaAs)
Useful Life 5 years minimum

OPERATING SPEED (See Note 2)
$\frac{12K\ CTS\ sec}{Counts\ per\ 360°} \times 60$ = RPM maximum

ENVIRONMENTAL

TEMPERATURE
Operating 0°C to +71°C
Storage −25°C to +85°C

VIBRATION
5 to 2000 HZ at 20 G s

SHOCK
50 G s for 11 MS duration

NOTES:

1 Slewing speed is the maximum rpm to which the encoder may be subjected without permanent degradation of performance

2 Operating speed is the maximum rpm at which the encoder may be read while maintaining full accuracy. In cases where operating speed calculates to a higher rpm than slewing speed, slewing speed becomes the limiting factor

3 Units with a lower starting torque can be provided by use of extra low torque bearings or when a reduced shaft loading capability is indicated

4 Heavy duty version uses larger sealed bearings and shaft per Fig 4

5 For count direction select option use configuration dash 7. Control is on Pin L
For CW increasing count connect Pin L to ground
For CCW increasing count, connect Pin L to +5 0 VDC

INPUT/OUTPUT CONNECTIONS

CONNECTOR PT07C 14 19P PIN NO	GRAY CODE OUTPUT	NATURAL BINARY OUTPUT	8421 BCD OUTPUT
A	G 0	2^0	1
B	G 1	2^1	2
C	G 2	[illegible]	4
D	G 3	[illegible]	[illegible]
E	G 4	2^4	10
F	G 5	[illegible]	20
G	G 6	[illegible]	[illegible]
H	G 7	[illegible]	[illegible]
J	G 8	[illegible]	[illegible]

CONNECTOR PT07C 14 19P PIN NO	GRAY CODE OUTPUT	NATURAL BINARY OUTPUT	8421 BCD OUTPUT
K	G9	[illegible]	200
L	DIRECTION CONTROL (-7 only)		
M	—	—	400
N	—	—	800
V	+5 0 ± 2% VDC		
T	GROUND		
S	CASE GROUND		
OTHERS	SPARE		

BEI Industrial Encoder Division
BEI MOTION SYSTEMS COMPANY
7230 Hollister Avenue • Goleta, California 93117-2891

Specification

924 - 02004 - 001

General Specifications

Type L25

Incremental Optical Encoder

ACTUAL SIZE

Notice: The design and specifications of the instruments and accessories illustrated and described in this publication are subject to improvement without notice.

REV	DESCRIPTION	DATE		
E	Per ECN 1831		PREP BY D. McGuire 8/18/77	02004-001
D	Per ECN 1613	1-17-83	CHK D. LaPlante	SHEET 1 OF 9
C	Updated per ECN 1372		APPD Jerry E. Jandt	REV D

BEI Industrial Encoder Division
BEI MOTION SYSTEMS COMPANY
7230 Hollister Avenue • Goleta, California 93117-2891

TITLE	NO.	Rev
General Specifications Type L25 Incremental Optical Encoder	02004-001 Sht 2	D

SPECIFICATIONS

1.0 Scope: This specification describes the BEI Industrial Encoder Division Low Torque, Instrument Grade Type L25 Incremental Optical Encoder.

2.0 Mechanical Specifications

2.1	Dimensions	See Figure 2
2.2	Shaft Diameter	.2497/.2495 Dia.
2.3	Optional Flat on Shaft	.50 long X .03 deep
2.4	Shaft Loading	Up to 5 lbs Axial and 8 lbs Radial
2.5	Shaft Runout	.0005 T.I.R. Max.
2.6	Starting Torque at 25°C	0.07 Oz. -In. Max.
2.7	Starting Torque at 25°C (With optional sealed bearings)	1.0 Oz. -In. Max.
2.8	Bearings	Class ABEC 7
2.9	Shaft	416 Stainless Steel
2.10	Housing	Die Cast Aluminum
2.11	Cover	Drawn Aluminum, .060" Wall
2.12	Bearing Life (mfr's specifications)	1 X 10^9 Revs at rated shaft loading
2.13	Moment of Inertia	4.1 X 10^{-4} Oz. In. Sec.^{2}
2.14	Slew Speed	5000 RPM Max.
2.15	Weight	13 Oz. Typ.

BEI Industrial Encoder Division
BEI MOTION SYSTEMS COMPANY
7230 Hollister Avenue • Goleta, California 93117-2891

TITLE	NO.	Rev
General Specifications Type L2S Incremental Optical Encoder	02004-001 Sht 3	D

3.0	Electrical Specifications	
3.1	Code	Incremental
3.2	Counts Per Shaft Turn	1 to 2S40
3.3	Supply Voltage	See Table I
3.4	Current Requirements	TTL 200 Ma Max, 1S0 Ma Typ CMOS 1S0 Ma Max, 12S Ma Typ
3.5	Output Format	2 Channels (A and B) in quadrature ± 27° electrical at 10 Khz See Figure I
3.6	Output Format Options	Index and Complementary outputs are available
3.7	Output Options	See Table I

TABLE I

I.C. Number	Type	Feature	Optional Pull-up Resistor	Output	Supply Voltage ± 5%
SN7404	T^2L	Totem Pole		16 MA/SV	+5 VDC
SN7406	T^2L	Open Collector Hi-Voltage	470 Ohms	40 MA/30V	+S VDC
SN74C04	CMOS				S to 1S VDC*
MC680	HTL	Totem Pole			15 VDC
MC681	HTL	Open Collector	1SK Ohms		1S VDC
MC689	HTL	Open Collector Hi-Voltage	1SK Ohms	20V	1S VDC
DM8830	T^2L	Line Driver			S VDC
MM88C30	CMOS	Line Driver			S to 15 VDC*

*Specify actual Voltage

BEI Industrial Encoder Division
BEI MOTION SYSTEMS COMPANY
7230 Hollister Avenue • Goleta, California 93117-2891

TITLE	NO.	Rev
General Specifications Type L25 Incremental Optical Encoder	02004-001	Rev
	Sht 4	0

3.8	Illumination	Incandescent Lamp (40,000 hours life) LED, Optional (Index up to 1270 CPT only)
3.9	Frequency Response (Channels A and B)	100 kHz
3.10	Frequency Response (Index)	100 kHz

4.0 Environmental Specifications

4.1	Temperature Operating Storage	 0 to 70°C Standard -25 to 90°C
4.2	Shock	50 G's for 11 MSEC duration
4.3	Vibration	5 to 2000 HZ @ 20 G's
4.4	Humidity	98% RH without condensation

5.0 Options - For the following option capabilities, consult factory for complete specifications.

5.1	Direction Sensing	Pulse Output X1, X2 or X4
5.2	Interpolation	Multiplied squarewave output X5
5.3	Dual Resolution	Selectable Output
5.4	Sinewave	Differential amplified outputs

BEI Industrial Encoder Division
BEI MOTION SYSTEMS COMPANY
7230 Hollister Avenue • Goleta, California 93117-2891

TITLE	NO.	Rev
General Specifications Type L25 Incremental Optical Encoder	02004-001 Sht 5	0

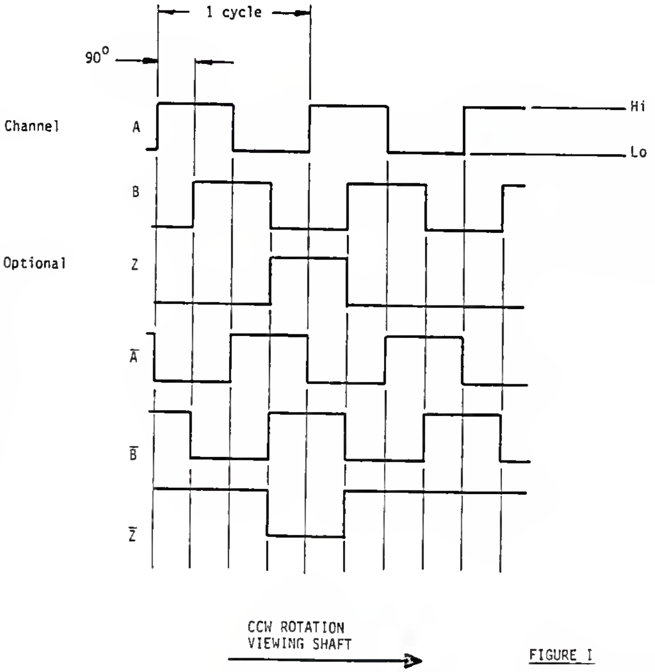
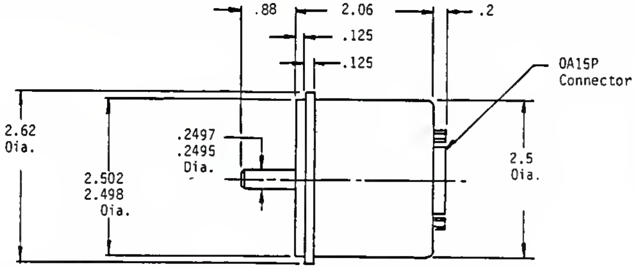


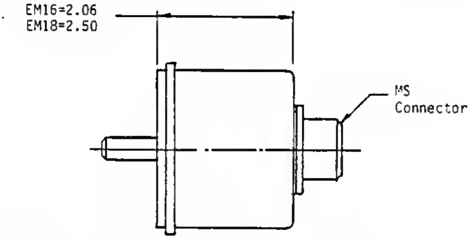
FIGURE I
OUTPUT WAVE FORMS

BEI Industrial Encoder Division
BEI MOTION SYSTEMS COMPANY
7230 Hollister Avenue • Goleta, California 93117-2891

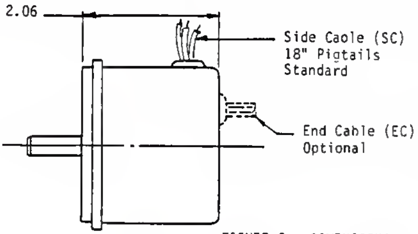
TITLE	NO.	Rev
General Specifications Type L25 Incremental Optical Encoder	02004-001 Sht 6	0



L25G....E015



L25G.....EM16 or EM18 Depending on Number of Outputs (See Table II)



L25G-----SC18 or EC18

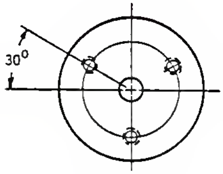
FIGURE 2 - DIMENSIONS

BEI Industrial Encoder Division
BEI MOTION SYSTEMS COMPANY
7230 Hollister Avenue • Goleta, California 93117-2891

TITLE	NO.	Rev
General Specifications Type L25 Incremental Optical Encoder	02004-001 Sht 7	0

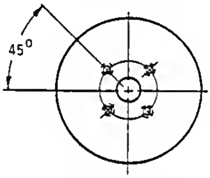
FIGURE 3

Face Mount Options



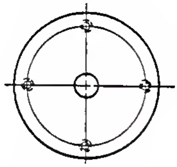
F1

10-32 UNF-2B
.188 Min. Deep
3 places equally spaced on a
1.875 Dia. bolt circle.



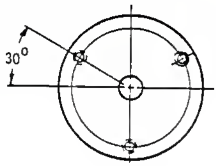
F2

4-40 UNC-2B
.250 Min. Deep
4 places equally spaced
on a 1.272 Dia. bolt circle
(.900 square, Ref)



F3

4-40 UNC-2B
.250 Min. deep
4 places equally spaced
on a 2.000 Dia. bolt circle



F4

6-32 UNC-2B
.250 Min. deep
3 places equally spaced
on a 2.000 Dia. bolt circle

BEI Industrial Encoder Division
BEI MOTION SYSTEMS COMPANY
7230 Hollister Avenue • Goleta, California 93117-2891

TITLE	NO.	Rev
General Specifications Type L25 Incremental Optical Encoder	02004-001	
	Sht B	D

TABLE II

OUTPUT TERMINATIONS

MS3102E-16S-1P				MS3102E-18-1P	
Output Option	Channels A, B and Z	Ch. A & B with Complements	Ch. A & Z with Complements	Pin	Ch. A,B & Z with Complements
Pin A	Channel A	A	A	A	A
B	B	B	$\bar{A}$	B	B
C	Z	$\bar{A}$	Z	C	Z
D	+V	+V	+V	D	+V
E	No Conn.	$\bar{B}$	$\bar{Z}$	E	No Conn.
F	Ground	Ground	Ground	F	Ground
G	Case Ground	Case Ground	Case Ground	G	Case Ground
				H	$\bar{A}$
				I	$\bar{B}$
				J	$\bar{Z}$

WIRE OR OA15P CONNECTOR TERMINATION

Function	Wire Color	OA15P Pin Number
Channel A	Yellow	13
B	Blue	14
Z	Orange	15
$\bar{A}$	White-Yellow	10
$\bar{B}$	White-Blue	11
$\bar{Z}$	White-Orange	12
+5V	Red	6
Ground	Black	1
Case Ground	Green	9

BEI Industrial Encoder Division
BEI MOTION SYSTEMS COMPANY
7230 Hollister Avenue • Goleta, California 93117-2891

TITLE	NO.	Rev
General Specifications Type L25 Incremental Optical Encoder	D2004-001 Sht 9	D

6.0 Ordering Information: Encoder may be specified using the following model numbering system:

Field	Box
TYPE: L = Light Duty	L
BASIC SIZE: 25 = 2.500	25
HOUSING CONFIGURATION LETTER: G = 2.62 Dia Servo Mount (Fig. 2)	G
—	—
FACE MOUNT OPTIONS (Fig. 3) F1, F2 or F3 Blank = None	
—	—
SHAFT SEAL CONFIGURATION: SB = Seal Integral with Bearing Blank = Shielded Bearing	
—	—
CYCLES PER TURN: Enter Cycles: 500 = 500 cycles 2500 = 2500 cycles Etc.	
—	—
NO. OF CHANNELS: A = Single Channel AB = Dual Quadrature Channels ABZ = Dual with Index AZ = Single with Index	
COMPLEMENTS: C = Complementary Outputs Blank = None	
—	—
OUTPUT I.C. 7406, 8830, 7404, 88C30, etc (See Table I) Followed by "R" = Pull-up Resistor	
—	—
ILLUMINATION: Blank = Incandescent (Standard) LED = Light Emitting Diode (Optional)	
—	—
OUTPUT TERMINATION LOCATION: E = End S = Side (Pigtail only)	
OUTPUT TERMINATION: M16 = MS3102E16S-1P Connector M18 = MS3102E18-1P Connector D15 = DA15P C = Pigtail cable followed by length, i.e. C18 = Pigtail cable 18" long	
—	—
S = Special Non-Standard Features specified on purchase order or customer's spec	

DEVELOPMENT OF DATA ACQUISITION AND CONTROL
FACILITIES FOR THE OPTIMIZATION OF
DRIVE LINE EFFICIENCY

by

KENT DOUGLAS FUNK

B.S., KANSAS STATE UNIVERSITY, 1984

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Mechanical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

ABSTRACT

Due to escalating fuel costs and increased capital costs associated with operating and owning agricultural tractors, considerable research has been conducted to improve the fuel efficiency and work rate of these units. The primary focus of this research has been to accurately define field load variations and to optimize engine power utilization. The potential savings from tractor performance optimization depends upon several factors; load variability, power level, engine characteristics, transmission characteristics, and tractive efficiency.

A review of previous engine optimization work has identified several limitations which may lead to non-optimal solutions. These limitations can be reduced by; using closed loop drive line controls, a transmission which has a large number of discrete ratios or a continuously variable transmission, and developing optimization algorithms which consider the entire drive line rather than focusing only on the engine.

A joint study of Computer Control of Agricultural Tractor Drive Lines was initiated in April, 1984, between the Agricultural Engineering and Mechanical Engineering Departments at Kansas State University. The objective of this effort is to develop and test a computer control system for optimizing the performance of a diesel engine and a continuously variable transmission as applied in an agricultural tractor. One of the primary tasks of this study is to develop laboratory facilities in order

to study drive line efficiency.

In order to fulfill the project's computer needs, two systems were developed. One system uses an ADAC 1000 data acquisition computer and is responsible for the supervisory functions, data recording, sub-system controls, and conversion of all analog data into digital forms. The other system uses a Motorola MC68000ECB single board computer and is responsible for drive line control and optimization.

The work completed on the ADAC 1000 falls into two main categories; a complete system upgrade, and development of a large software package. The structure of the software package is based upon concepts used in concurrent programming in order to preserve real time capabilities.

The work completed on the MC68000ECB includes both hardware and software developments. The hardware developments include bus expansion buffering, I/O expansion, optical isolation, and digital interfacing to numerous external devices. The software developed provides the framework for all future developments. Currently, the software executes a fully interactive environment which is useful for drive line mapping.

This project is on-going with the test facility completed, performance data collected, and a basic optimization algorithm outlined. Plans for future work include: analyzing the collected data to establish relationships between control inputs and drive line outputs, and developing a computer simulation of the optimization algorithm in order to evaluate dynamic and performance considerations.